UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

C++ RPG Terrains

Llibreria de
generació procedural de terrenys
a temps real
en C++

Treball de final de grau

Grau en disseny i desenvolupament de videojocs

Alumne: Hernàndez Làzaro, David

Pla 2014

Director: Lluch-Ariet, Magí

RESUM

Aquest document correspon a la descripció del treball pel desenvolupament de la llibreria “Realtime Procedurally Generated Terrains in C++” (**C++ RPG Terrains**). Es tracta d'una llibreria per a generar i renderitzar terrenys 3D virtualment infinits per a videojocs a temps real i de manera dinàmica.

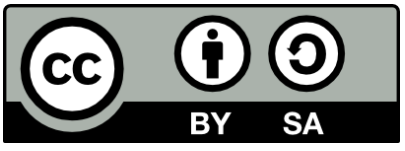
El resultat obtingut és una llibreria personalitzable que pot ser inclosa en projectes fets en C++ de manera senzilla. Per a provar la seva utilitat en l'entorn dels videojocs, aquest TFG inclou la creació d'un petit joc utilitzant la llibreria.

En aquest document es descriuen les tècniques de generació procedural de terrenys, de render i d'optimització necessàries per a obtenir el resultat especificat i les dificultats trobades al llarg del desenvolupament.

El resultat d’aquest projecte, C++ RPG Terrains, és públic sota llicència GNU (General Public License)(29) al repositori de Github “V_Terrains”(32), el nom previ que va rebre el projecte.

Paraules clau

Procedural, Terrenys, Programació, C++, GLSL, Videojocs, Heightmap



Índex

Índex de taules

Índex de figures

1. Introducció

1.1 Contingut del treball

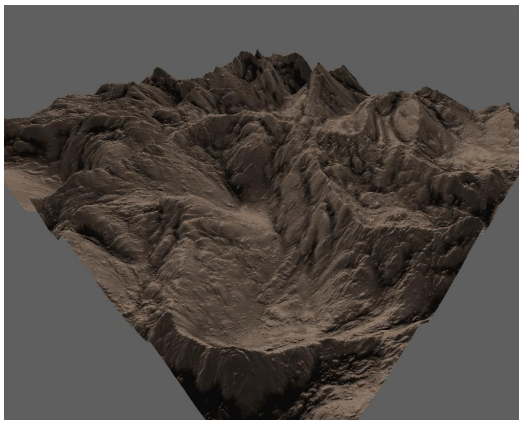
En aquest treball s'analitza com generar terrenys procedurals en temps real per a videojocs i les optimitzacions necessàries per a aconseguir-ho, i com es genera una llibreria en C++ recollint els mètodes investigats.

En videojocs, els terrenys són el terra sobre que es camina, sobretot en espais oberts i entorns grans. Són molt similars a les malles de qualsevol altre objecte 3D, com a mínim en aparença dins l'aplicació, però el seu funcionament intern és completament diferent, doncs requereix moltes optimitzacions per poder mantenir una extensió tan grossa carregada i evitar que afecti el rendiment.

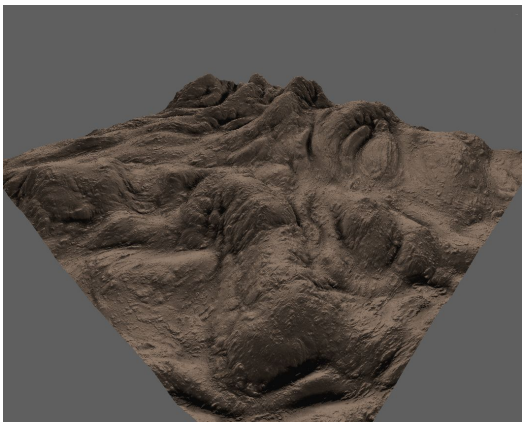
En general, el contingut procedural, és aquell generat directament per l'ordinador a través d'un algoritme.

Per tant, en parlar de terrenys procedurals, ens referim a un terreny generat automàticament, sense la intervenció directa d'una persona, de manera automàtica i dinàmica, a temps real, per a donar la sensació d'un terreny infinit al jugador.

Les figures inferiors, 1.1 i 1.2, mostren dos possibles resultats de terreny procedural generats a partir d'un mateix algoritme. Això són "chunks", o "fragments del terreny", que es col·locaran un al costat de l'altre per a formar una extensió gran de terreny que es pot generar per peces dinàmicament.



F 1.1 - Exemple de terreny procedural 1



F 1.2 - Exemple de terreny procedural 2

1.2 Motivació

Curiositat. Aprenentatge. Reptes.

Aquestes són realment les motivacions que em duen a investigar i treballar en aquest tema, com podrien ser molts altres. Treballant en el món dels videojocs, a cada moment es descobreixen noves tecnologies que canvien la manera de fer jocs, cada una més complexa e interessant que l'anterior. I tant de bo les pogués conèixer totes.

Tot i que... per què aquest tema en concret? Bé, en l'àmbit personal, sempre he sentit fascinació pel contingut procedural i la generació automàtica d'entorns i nivells i la capacitat de, amb relatiu poc esforç, obtenir contingut virtualment infinit.

Cada vegada més jocs aprofiten aquesta tecnologia, en major o menor mesura, per a poder agilitzar el seu procediment de creació de contingut, tot i que el seu potencial probablement encara no s'està utilitzant al màxim.

I, per què terrenys? Doncs, és una àrea en la qual ja he treballat prèviament en altres projectes, com RiKarts(15), tot i que no vaig poder dedicar-hi tant temps que hauria volgut per a sentir-me realment satisfet amb el resultat, el que em motiva a seguir-hi treballant i investigant. És una àrea que lliga especialment bé amb el contingut procedural, i sobre la qual ja existeix la tecnologia necessària per a desenvolupar-la, però per algun motiu poques llibreries que facilitin la feina de creació de terrenys. Com a mínim sense utilitzar engines ja creats o aplicacions externes al programari.

A més a més, ja no sols la llibreria com a complement al codi d'un projecte, sinó un exemple de com aproximar-se als reptes que puguin aparèixer a l'hora de generar terrenys procedurals, pot ajudar i/o motivar a petits estudis de videojocs que no tindrien la capacitat o el temps d'incloure features com aquestes als seus projectes a considerar fer-ho.

1.3 Formulació del problema

La generació de terrenys procedurals no és un desconegut per als videojocs: ja existeixen nombrosos papers que en parlen(4,7,11) i engines i aplicacions que faciliten la seva obtenció, o s'hi acosten. Tot i això, com s'especifica més endavant en el "State of the Art"(2.3), totes aquestes implementacions no funcionen per a aplicacions standalone o no permeten la generació de contingut de manera purament procedural, requerint l'edició i refinament del resultat per part d'una persona.

Aquest és justament el problema: no hi ha una eina que permeti la creació de terrenys purament procedurals de manera ràpida i eficient, fora d'engines tancats.
Tot el codi que crearem, tota la tecnologia que usarem i el resultat que obtindrem ja s'han aconseguit prèviament, són fites que ja formen part d'alguns jocs existents, però no de manera oberta i accessible per a tothom.

1.4 Objectius generals

L'objectiu d'aquest treball és desenvolupar una llibreria pública i oberta en C++ que serveixi tant com a eina per a generar terrenys de manera eficient i ràpida, o com a referència, guia o framework a utilitzar a l'hora de generar terrenys procedurals per a qualsevol joc o aplicació.

Objectius del treball

O1. Dissenyar i desenvolupar una llibreria per la generació dinàmica dels terrenys

O2. Permetre a l'usuari personalitzar el funcionament de la llibreria

O3. Aportar les eines necessàries per a incloure al terreny textures definides per l'usuari

O4. Publicar la llibreria de manera oberta, sota la llicència GNU General Public License(29).

O5. Analitzar l'eficiència i rendiment dels mètodes emprats per a generar i renderitzar terrenys

O6. Crear una petita aplicació que utilitzi la llibreria per demostrar la seva utilitat

En complir tots aquests objectius, el resultat hauria de ser una llibreria amb les funcionalitats descrites a la taula 1.1.

Configurable i readaptable	
Descripció	L'usuari pot substituir les funcions de la llibreria per les seves pròpies, així com injectar codi per a personalitzar el funcionament i el resultat. Accés a les variables que permeten modificar el terreny de manera ràpida i senzilla.
Conseqüències	El desenvolupador tindrà accés a variables públiques que permetran modificar: la generació del heightmap, creació de la malla, render i límits de memòria i CPU consumits; a més a més dels punters a les funcions utilitzades per a poder injectar el codi propi.
Infinit	
Descripció	La generació del terreny es produeix en temps d'execució, generant i alliberant els recursos a conveniència per a eliminar virtualment els límits de la zona explorable.
Conseqüències	L'usuari de l'aplicació que integri aquesta llibreria percebrà el terreny com a infinit. Les vores d'aquest no seran visibles per molt desplaçament que faci.
Repetible	
Descripció	Dues configuracions idèntiques donaran exactament el mateix resultat, per a poder recuperar qualsevol terreny generat guardant únicament la configuració, i evitar així la necessitat de guardar cap informació del terreny generat.
Conseqüències	Un usuari que revisiti una localització prèviament visitada, trobarà exactament el mateix terreny que en l'anterior visita, encara que el desenvolupador no emmagatzemi cap dada sobre aquella secció en concret.
Purament procedural	
Descripció	El terreny no necessitarà la intervenció de cap persona després de ser generat per a ser utilitzable dins un joc.
Conseqüències	El desenvolupador podrà relegar qualsevol tasca relacionada amb la generació del terreny al codi i no necessitarà fer-hi modificacions post-generació.
Òptim, a temps real	
Descripció	Capaç de funcionar paral·lelament a la resta del programa, amb impacte mínim, aplicant les optimitzacions necessàries per a aconseguir-ho.
Conseqüències	El desenvolupador no veurà alentit el seu programari (de manera notable) a causa de la llibreria.
Render final	
Descripció	Qualitat de render suficient per a formar part d'un projecte sense modificacions.
Conseqüències	El desenvolupador podria utilitzar el render proporcionat en un projecte final sense que aparegués incomplet.

T 1.1 - Funcionalitats

1.5 Objectius específics

A nivell més específic, tant en qualitat lectiva com per al correcte assoliment dels objectius mencionats prèviament, es llisten algunes de les tecnologies específiques i possibles reptes que s'investiguen al llarg del desenvolupament.

1.5.1 Generació del terreny

OE1.1 Generar heightmaps: la base per a qualsevol terreny 3D en superfície és un heightmap. En aquest cas, la generació de heightmaps infinits i repetibles, interessants visualment i configurables, amb diferents nivells de detall.

OE1.2 Generar malles: crear malles 3D a partir dels heightmaps de manera eficient i ràpida.

OE1.3 Calcular les textures i normals: Calcular la distribució de les diferents textures sobre el terreny i les normals per a la posterior il·luminació.

OE1.4 Generar LODs, stitching: Generar diferents nivells de detall de la malla i generar un *algoritme* de "sitching" que eviti l'aparició de forats en aquesta.

1.5.2 Render

OE2.1 Renderitzar meshes: Dibuixar per pantalla el terreny generat, aplicant llums i altres efectes de millora.

OE2.2 Càlcular posicions i fades entre textures: fade entre textures i processament dels diferents mapes, com el de normals.

OE2.3 Aplicar efectes de postprocessament: Efectes de boira, blur i/o aigua.

1.5.3 Optimització real-time

OE3.1 Dividir la càrrega entre CPU i GPU: utilitzar shaders i la potència de la GPU per a augmentar l'eficiència de la generació.

OE3.2 Gestionar "chunks" o fragments de terreny: *Algoritme* encarregat de la generació de nou terreny, destrucció de l'antic i d'assignar el LOD adequat per a donar la sensació de terreny infinit i dissimular la generació del nou.

OE3.3 Aplicar multithreading: Distribució de la càrrega en diferents "threads" d'execució per a repartir el pes i evitar l'alentiment del programa.

OE3.4 Precarregar "chunks": detectar els moments en què no s'està utilitzant tota la potència del processador per a carregar zones del terreny encara no visibles de manera previsória per a facilitar la seva posterior generació.

1.5.4 Aplicació vinculada a la llibreria

OE4.1 Aplicació interactiva: desenvolupar un joc simple per a explorar les capacitats de la llibreria de manera interactiva i consolidar i testejar les seves funcionalitats.

1.6 Beneficiaris

I tot això, per què?

Primer de tot, la utilització de C++ RPG Terrains pot ser útil per a qualsevol desenvolupador que necessiti generar una aplicació amb terrenys d'aquestes característiques. Petites empreses, estudiants, gent que s'inicia dins el món dels jocs o simuladors, doncs una de les utilitats més comunes d'aquest tipus de tecnologia és i ha estat la dels simuladors de vol.

Utilitzar una llibreria com aquesta no únicament estalvia temps: redueix els costos de producció, obre noves possibilitats que possiblement no s'havien considerat anteriorment i, en general, permet destinar els recursos i els esforços a no haver de retallar features en la producció o a polir el producte i obtenir una qualitat millor en general.

I, en definitiva, a permetre al jugador o usuari una millor experiència a l'utilitzar qualsevol aplicació en què aquesta tecnologia que desenvoluparem sigui utilitzada.

I, també, per a mi mateix. Com a experiència, com a creixement personal, com a ampliació dels coneixements, com a repte.

2. Estat de l'art

2.1 Terrenys

Hi ha diferents tipus de terrenys comunament usats en videojocs, agrupats per la manera en què emmagatzemen la informació dels vèrtexs de la malla. Els mètodes més comuns són, per a malles 2D, els Heightmaps, descrits a la taula 2.1; per a malles 3D, els Voxels, descrits a la taula 2.2; i per a casos més específics, les malles 3D modelades, descrites a la taula 2.3.

2.1.1 Heightmaps	
En els heightmaps únicament s'emmagatzema l'alçada (y) de cada un dels vèrtexs, normalment en una textura 2D, en què cada píxel correspon a un dels vèrtexs. El terreny es genera creant una geometria a partir d'aquesta informació o utilitzant els shaders de geometria.	
Aquest mètode és l'utilitzat en aquest projecte, ja que ofereix un balanç correcte entre possibilitats, complexitat i eficiència, i és especialment adequat a l'hora de generar soroll procedural.	
Pros Utilitzen poca memòria, ja que es guarda un únic valor per vèrtex. És un dels mètodes més ràpids de renderitzar terreny. Facilitat per a generar LODs de manera automàtica. Creació de heightmaps és molt simple. Els motors de física treballen de manera molt eficient amb ells.	**Cons** Poc detall. S'ha de suplir aplicant filtres de soroll o "detail maps". La precisió no pot ser major que la resolució de la textura 2D. Només una alçada per x/y. No poden haver-hi coves o volades.
Ha estat utilitzat en jocs com: "Far Cry 4"(2014), "Cities, Skylines"(2015)	

T 2.1 - Heightmaps

2.1.2 Voxels

Similar als heightmaps, però emmagatzemant dades en una matriu de tres dimensions. L'exemple més clar i visual és el joc "Minecraft", en què cada cub es podria considerar un "voxel".

Aquesta tècnica es considera "out of scope" d'aquest projecte, puix que el cost i la variabilitat són massa alts.

Pros	**Cons**
Informació 3D continua, que permet tenir informació d'elements no visibles, com vetes de minerals. Les dades sense comprimir són fàcils de modificar, fins i tot in-game. Permet crear formes complexes en terrenys, com coves i volades. Permet crear la varietat més gran de terrenys de manera relativament senzilla.	Són lents de renderitzar i de processar. Emmagatzemar les dades necessàries ocupa molta memòria, fins i tot comprimint-les utilitzant octrees. Comprimir les dades complica molt el seu ús.

Ha estat utilitzat en jocs com “Minecraft”(2009) o “No Man’s Sky”(2016)

T 2.2 - Voxels

2.1.3 Meshes

En aquest cas el terreny és una mesh creada manualment en un programa extern.

No utilitzem aquesta tècnica per a aquest treball, perquè una implementació procedural d'aquest sistema és massa complexa i no acostuma a utilitzar-se en videojocs.

Pros	**Cons**
Els més ràpids de renderitzar. Precisos i eficients, ja que es pot afegir detall i polígons allà on sigui necessari manualment. Requereixen menys memòria que els altres mètodes. Sense artefactes visuals, ja que no es modifica la mesh. Si està ben generada, no donarà problemes. Llibertat per a crear qualsevol tipus de forma.	Dificultat de crear LODs, a no ser que es tinguin diverses meshes. Difícils de modificar. Poc eficients a l’hora de crear-los i generar-los, havent de fer-los manualment.

És difícil trobar usos actuals a aquesta tècnica fora d'entorns molt específics, però sí que va ser molt usada en jocs més antics per la manca d'habilitat de calcular el terreny en temps d'execució, com Zelda o Mario.

T 2.3 - Meshes

No és estrany veure barreges de diverses tècniques o aplicar noves implementacions per a suplir les mancances d'alguns dels mètodes anteriors. Un dels més comuns és barrejar heightmaps amb geometria esculpida manualment per a simular volades i formes geomètriques més complexes.

Aquest projecte es podria arribar a ampliar per a utilitzar aquesta tècnica, combinant el heightmap amb voxels per a coves o geometria predefinida. Tot i això, queda "out of scope" d'aquest projecte i no es tindrà en compte.

Existeixen també altres mètodes menys comuns, com ara generar terrenys únicament usant els shaders, però el jugador no pot tenir cap mena d'interacció amb aquest terreny i es perd control sobre el resultat, doncs tots els càlculs es fan a la targeta gràfica i la memòria no hi té accés. Poden ser molt eficients, però sols són útils en casos molt concrets.

2.2 Generació procedural de terrenys

2.2.1 Introducció al contingut procedural

La generació de contingut procedural (o PCG) en videojocs es refereix a la creació de nou contingut a través d'un algoritme. Tot i que, tal com es comenta a la introducció de "What is Procedural content generation? Mario on the borderline"(2), aquesta definició és molt àmplia, i tot i això segueix sent la millor que tenim fins al moment, doncs la varietat d'usos d'aquest contingut és tant extensa com variada.

La utilització de PCG s'aplica habitualment a:

- Aleatoritzar el contingut per a oferir rejugabilitat. Els nivells són generats proceduralment per un algoritme cada vegada que es comença una partida, de manera que el contingut és virtualment infinit. Han utilitzat aquesta tècnica jocs com “Rogue”(1980), “Diablo” (1996) o la generació d’armes de Borderlands(2009).

- Reduir la memòria i els recursos necessaris. El contingut del joc no varia d’una sessió a una altra, però la generació d’aquest es pot fer a través d’un algoritme. Es poden veure exemples en jocs com The Sentinel”(1985) i el recent “No man’s sky”(2016), que ofereix 18.000.000 trilions de planetes en tan sols 6GB de joc (dels quals, la majoria són sons).

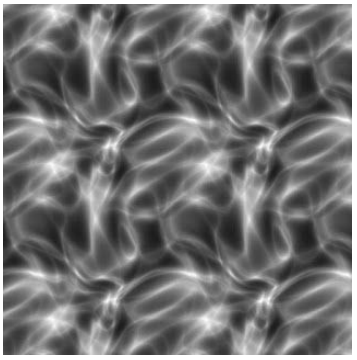
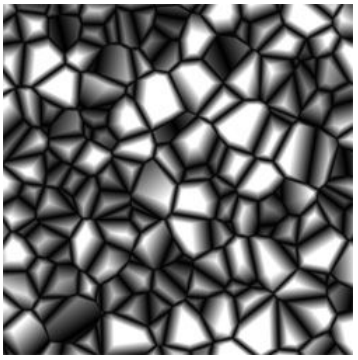
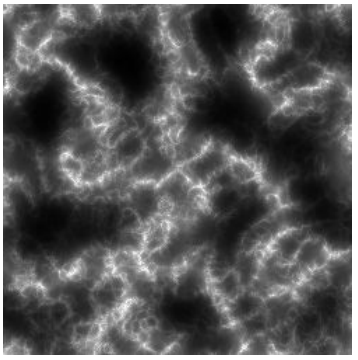
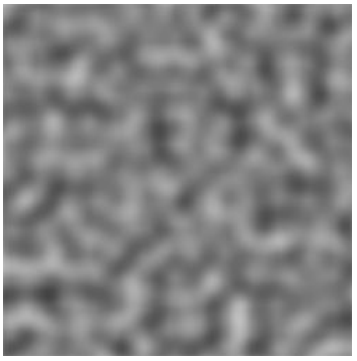
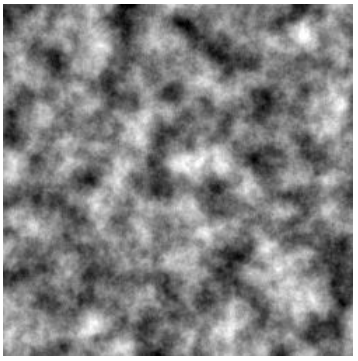
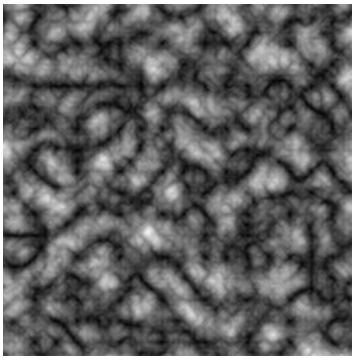
- Generar detall i contingut addicional per a poblar el joc, per a estalviar feina als dissenyadors, com col·locar els brins d’herba, generar els arbres d’un bosc o l’erosió i rascades d’una textura. Un exemple és l’eina “SpeedTree”(2003) (30) per a generar varietat d’arbres.

El primer contingut que es considera generat de manera procedural, data de l’any 1952, per Alan Turing. Va fer M.U.C. (Manchester University Computer) per a provar que els ordinadors podien treballar amb paraules a més a més de nombres, i va generar cartes d’amor procedurals.

2.2.2 Generació de soroll

Tots els terrenys acostumen a començar amb una generació de soroll sobre una textura de manera pseudo-aleatoria. Utilitzem el prefix “pseudo” perquè, si bé el resultat és aleatori, s’utilitza una llavor concreta per a generar el soroll, i aplicant una mateixa llavor múltiples vegades el resultat hauria de repetir-se.

Existeixen diferents tipus de soroll, tots amb variants, beneficis i inconvenients, i els resultats obtinguts són dràsticament diferents. Alguns exemples poden ser vistos a la taula 2.4.

Caustic Noise	**Worley Voronoi Noise**	**Fractional Brownian Motion Noise**
		
Usat en animacions d’aigua o efectes sub-aquatics	Sense aplicacions habituals	Utilitzat en textures d’electricitat o fluïds.
Regular Perlin Noise	**Fractal Perlin Noise**	**Turbulence Perlin Noise**
		
Utilitzar com a filtre en aplicacions d’edició d’imatge.	Utilitzat en la creació de terrenys i heightmaps.	Sense aplicacions habituals

T 2.4 - Tipus de soroll

El soroll més empleat per a videojocs és el "Fractal Perlin Noise" (13). Primer de tot, perquè el resultat que dóna és el més similar a un terreny, amb els ajustaments necessaris, perquè es tracta d'un soroll generat punt a punt, que permet crear textures "tilejables" i un pla infinit de soroll, a més a més de ser fàcilment adaptable i configurable.

Per a una explicació més en detall sobre aquest tipus de soroll, anar a aquest enllaç, Understanding perlin noise - Adrian’s soapbox (20).

2.2.3 Simulació d'erosió

El següent pas acostuma a ser la simulació d'erosió: del vent, de l'aigua, dels elements en general, per a obtenir un terreny més realista i menys aleatori. Això inclou la simulació d'oceans, llacs i rius, a més a més de modificar el heightmap de manera adequada per a simular el pas del temps i la naturalitat del terreny.

Aquest pas es veurà molt simplificat dins aquest projecte, per qüestió de temps, limitant-se a la generació d'oceans en les zones més baixes.

Per a més informació i exemples sobre algoritmes d'erosió, veure l'obra de Jacob Olsen: *Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games* (5).

2.2.4 Render

A l'hora de renderitzar els terrenys trobem dues opcions principals, que diferencien sobretot a quina part del maquinari posen la càrrega de treball.

- **Render a partir de malla**

En aquest cas, la CPU és qui s'endú la major part de la feina.

A partir del heightmap, es genera una malla poligonal: per a cada píxel es genera un vèrtex a l'alçada corresponent i es calculen les normals i les UVs entre altres, i es prepara la malla per a poder-la renderitzar com qualsevol altre objecte 3D. Aquest procés requereix tenir una malla diferent per a cada heightmap, el que pot consumir força memòria.

Posteriorment, no és necessari un geometry shader, i s'apliquen les textures corresponents a través del píxel shader.

- **Render a partir de heightmap**

En aquest cas, la GPU és qui fa la feina.

S'envia el heightmap directament als shaders i una malla plana, sense elevacions. És el geometry shader qui deformarà la malla amb la informació rebuda del heightmap a cada frame. I, posteriorment, el píxel shader aplicarà les textures, de la mateixa manera que en l'alternativa.

A partir de malla	**A partir del heightmap**
Alt consum de CPU Necessitat d'alocatar molta memòria Generació més lenta	Alt consum de GPU Recalcular les elevacions a cada frame Necessita simplificar les operacions realitzades

2.3 Aplicacions similars

Existeixen aplicacions capaces de generar terrenys similars i s'agrupen en diverses categories.

2.3.1 Engines

Els engines més utilitzats, com Unity(23) o Unreal(24), ofereixen terrenys amb totes les optimitzacions necessàries per al seu correcte funcionament sobre els quals treballar, i eines per a editar-los manualment. Tot i això, tota la generació del soroll s'ha de fer manualment a través de codi, o buscar plug-ins o complements fets per altres usuaris, que s'encarreguin de la PCG.

Si bé és un dels camins més fàcils per a obtenir terrenys procedurals amb qualitat professional, el desenvolupador està obligat a casar-se amb un dels engines i utilitzar-lo com a eina de desenvolupament, de manera que podria no ser el que busquen creadors amb motors propis o que treballen directament en C++.

2.3.2 Aplicacions de generació de terreny

Existeixen també aplicacions independents que són exclusivament eines de generació de terreny, com ara World Creator(25), World Machine(26) o PicoGen(27). Aquests programes permeten la creació de terrenys d'una qualitat i detall increïbles, tot i que no són purament procedurals, requereixen l'acció de l'usuari. Són més similars a "editors" que no pas "generadors".

A més a més, generen regions limitades de terreny, no permeten crear terrenys infinits. I el resultat s'exporta com a malla 3D o mapes, de manera que el resultat obtingut del programa és estàtic i no pot variar un cop extret. A més a més, les optimitzacions a l'hora de renderitzar i els LODs encara han de ser executats per l'usuari a través del codi.

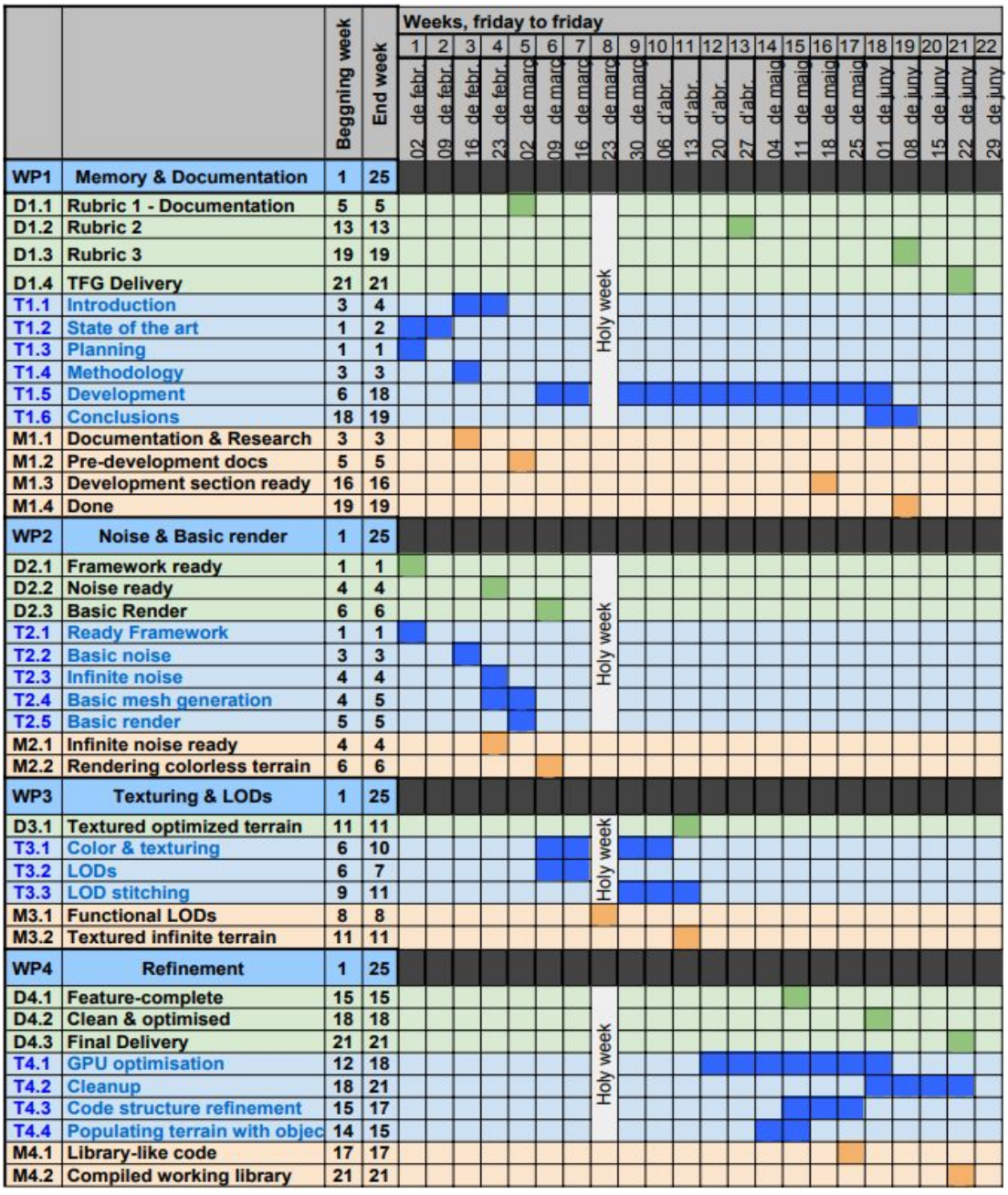
2.3.3 Generadors de soroll

També existeixen llibreries capaces de generar soroll, Perlin entre d'altres, o heightmaps a partir de geografia real, com Terrain Party(28) però l'usuari encara s'ha d'encarregar de tota la resta: des de generar i renderitzar el terreny a les optimitzacions de render. És l'alternativa més bàsica i a més baix nivell, sense contar fer el projecte des de 0.

3. Planificació

3.1 Planificació temporal

Per a distribuir les tasques al llarg del temps, he utilitzat un diagrama de Gantt que ens mostra el temps assignat previst per a cada una de les tasques i la seva distribució, així com les "Milestones" o fites a les que es vol arribar en certes dates. Aquestes es podran utilitzar per a mesurar la velocitat a la qual avança el projecte respecte al pla establert i ajustar els objectius en conseqüència. Tot això és representat gràficament al Gantt, taula 3.1.



T 3.1 - Gantt

3.2 Pla de costos

Tenim una segona taula, 3.2, dividida per tasques, en què es representa amb més exactitud el temps previst per a cada una de les tasques, el seu cost en temps i les tasques de què depenen cada una, de manera bloquejant.

WP1	Memory & Documentation	Milestone	Estimated hours	Potential deviation	Planned hours (with deviation)	Cost	Previous requisites
T1.1	Introduction	M1.2	1	None	1	9,00 €	
T1.4	State of the art	M1.1	6	Low	7,2	64,80 €	
T1.5	Planning	M1.1	5	Low	6	54,00 €	
T1.6	Methodology	M1.1	1	None	1	9,00 €	
T1.7	Development	M1.3	20	Average	28	252,00 €	
T1.8	Conclusions	M1.4	1,5	Very low	1,65	14,85 €	
	Total		34,5		44,85	403,65 €	Previous requisites
WP2	Noise & Basic render	Milestone	Estimated hours	Potential deviation	Planned hours (with deviation)	Cost	
T2.1	Ready Framework	M2.1	8	None	8	72,00 €	
T2.2	Basic noise	M2.1	10	Very low	11	99,00 €	T2.1
T2.3	Infinite noise	M2.1	8	Low	9,6	86,40 €	T2.2
T2.4	Basic mesh generation	M2.2	14	Low	16,8	151,20 €	T2.1
T2.5	Basic render	M2.2	10	Average	14	126,00 €	T2.4
	Total		50		59,4	534,60 €	Previous requisites
WP3	Texturing & LODs	Milestone	Estimated hours	Potential deviation	Planned hours (with deviation)	Cost	
T3.1	Color & texturing	M3.2	15	High	25,5	229,50 €	T2.5
T3.2	LODs	M3.1	20	Average	28	252,00 €	T2.4
T3.3	LOD stitching	M3.2	15	High	25,5	229,50 €	T3.2
	Total		50		79	711,00 €	Previous requisites
WP4	Refinement	Milestone	Estimated hours	Potential deviation	Planned hours (with deviation)	Cost	
T4.1	GPU optimisation	M4.2	25	Very high	50	450,00 €	T3.1
T4.2	Cleanup	M4.2	40	Low	48	432,00 €	
T4.3	Code structure refinement	M4.1	15	Average	21	189,00 €	
T4.4	Populating terrain with objects	M4.1	10	High	17	153,00 €	T2.4
	Total		90		136	1.224,00 €	
			224,5		319,25	2.873,25 €	

Potential deviation	Deviation applied
None	100%
Very low	110%
Low	120%
Average	140%
High	170%
Very high	200%

Cost per hour	9,00 €

T 3.2 - Tasques

A continuació, taula .3.3, que contempla els costos prevists per a la realització del projecte. Cal tenir en compte que aquest projecte no és "full-time", sinó que comprèn sols al voltant de 300 hores al llarg de 6 mesos, pel que la valoració d'alguns dels costos indirectes, com ara aigua i llum, s'intenta fer sols de la part proporcional al temps dedicat al projecte respecte al consum total.

Type	Subject	Price	Type	Years of amortization	Total price
Direct costs					
Personal	Salary / Time	2.873,25 €	Total		**2.873,25 €**
Equipment	Desk	150,00 €	Amortization	5	**12,50 €**
	Chair	100,00 €	Amortization	5	**8,33 €**
	Computer	1.000,00 €	Amortization	3	**138,89 €**
	Screen	110,00 €	Amortization	3	**15,28 €**
	Mouse	30,00 €	Amortization	3	**4,17 €**
	Keyboard	40,00 €	Amortization	3	**5,56 €**
Consumables	500 sheets of paper	3,00 €	Unique		**3,00 €**
	Pencils	3,00 €	Monthly		**15,00 €**
Software	Visual Studio	0,00 €	Monthly		**0,00 €**
	Github	7,00 €	Monthly		**35,00 €**
	HacknPlan	6,00 €	Monthly		**30,00 €**
Indirect costs					
Maintenance	Electricity	20,00 €	Monthly		**100,00 €**
	Water	12,00 €	Monthly		**60,00 €**
	Food & Snacks	30,00 €	Monthly		**150,00 €**
Total					**3.450,97 €**

Duration of project in months:
5

T 3.3 - Costos

Podem observar clarament com el principal cost és el temps invertit i el manteniment relacionat amb les necessitats bàsiques, que han estat aproximades. Això era d'esperar, ja que en projectes de programació i desenvolupament tecnològic, el material necessari és mínim i el software accessible normalment de manera gratuïta per als usuaris, o amb alternatives de baix cost.

3.3 DAFO

Per a obtenir una visió general dels riscos de la realització d'aquest projecte i les possibles complicacions que podrien aparèixer, així com els punts forts que és bo reforçar per a aconseguir que el projecte destaqui més i sigui més complet.

Debilitats	**Amenaces**
Projecte realitzat a temps parcial, sense plena dedicació. Sense molt coneixement previ, aprenentatge autònom. Estudiant, menys experiència en programar, el codi podria ser menys òptim.	Ja hi ha eines que aconsegueixen resultats similars de manera més eficient. El contingut procedural genèric pot no adaptar-se a les necessitats d'un projecte específic.
Fortaleses	**Oportunitats**
He treballat abans en terrenys i, encara que limitat, ja tinc coneixement dels problemes que es poden presentar. Flexibilitat i capacitat de prendre riscos i provar noves alternatives.	No hi ha cap eina que aconsegueixi aquest resultat en temps real, en C++ directament, independentment d'engines i software extern. Es tracta d'un treball d'estudiant, sense possibles pèrdues financeres, de manera que es poden prendre riscos sense repercussions greus.

T 3.4 - DAFO

3.4 Riscos i plans de contingències

3.4.1 Riscos durant el desenvolupament

Falta de temps

Un dels perills més comuns per a un projecte d'aquestes dimensions és errar en la planificació i abast del projecte, intentant abastar més del que es pot aconseguir o fent previsions de temps imprecises, el que pot dur a un projecte inacabat al final.

Per a evitar aquest problema, he:

- Aplicat un modificador a cada una de les estimacions de temps de les tasques, sempre a l'alça, per intentar minimitzar la possibilitat de falta de temps a l'hora de crear una feature.
- Aplicat els coneixements obtinguts en projectes anteriors, en què he treballat ja en terrenys, per a intentar tenir la mínima desviació quant a hores.
- Deixat un temps al final del projecte, en una tasca anomenada "CleanUp", que tant pot ser per a millorar el codi com per acabar i refinar features les quals el desenvolupament s'hagi allargat més del previst.
- Establert "Milestones" que permeten identificar l'estat del projecte en general i el progrés real contra el planificat, per a poder identificar el problema.

En cas que no es pugui prevenir:

- Es revisaran els objectius, ja que al ser "Feature Based", tal com s'explicarà en l'apartat de metodologia, es poden treure algunes features sense que afecti el resultat o la performance de la resta, com per exemple, poblar el terreny amb objectes.

Falta de coneixement

En treballar en noves tecnologies i creant codi nou, és possible que en algun moment no estigui clara la manera de continuar o la millor manera d'atansar-se a un problema. Una solució poc òptima podria significar haver de refer part de la feina feta més endavant.

Per a evitar aquest problema, he:

- Triat una àrea de coneixement en la que he treballat prèviament, per poder tenir una lleugera idea d'on podrien aparèixer complicacions.
- Investigat profundament l'estat de l'art i les diferents maneres i opcions de realitzar cada una de les features per a intentar fer-les de la millor manera possible.
- Assignat un espai de temps a cada una de les features i tasques per a "dissenyar i planificar", per a poder investigar individualment cada una d'elles abans d'executar-les.

En cas que no es pugui prevenir:

- De manera molt vinculada amb el risc anterior, no tenir el coneixement necessari va vinculat amb la pèrdua de temps, pel que s'apliquen les solucions del problema previ.

3.4.2 Riscos d'adaptabilitat

Un dels motius que el contingut procedural és poc present en llibreries i similars, és la gran dependència que té del disseny de l'aplicació. Cada joc, cada aplicació té unes necessitats molt específiques, i el gameplay i la diversió són l'element principal, i els continguts procedurals han de treballar conjuntament amb les mecàniques específiques per tal d'obtenir el millor resultat possible.

Project dependant

La llibreria no es pot adaptar suficientment als possibles usos que se li vulguin donar i, encara que compleixi amb els objectius i la funcionalitat sigui correcta, no és de valor a l'hora de crear videojocs utilitzant-la per a la dissociació entre el contingut procedural i el joc en si.

Per a evitar aquest problema, he:

- Fet molt incís en la personalització de la llibreria, oferint la possibilitat d'injectar codi propi i modificar el funcionament de parts majors del projecte per tal de poder readaptar-lo i fer que s'ajusti al màxim possible.
- Plantejat el projecte com a "open source" perquè, fins i tot en el cas que la llibreria no sigui utilitzable, el codi present i les tècniques usades es puguin utilitzar com a referència a l'hora de desenvolupar les aplicacions.

En cas que no es pugui prevenir:

- Es seguirà oferint el projecte com a open-source, i amb dades de contacte amb el desenvolupador, de manera que fins i tot en el cas que no es pugui aprofitar la llibreria, es puguin utilitzar parts del codi o resoldre dubtes.

3.4.3 Tasques de risc

A la taula T 3.2 - Tasques, s'identifiquen algunes tasques amb desviacions potencials mitges o altes. Aquestes són, probablement, les que poden ser considerades com de major risc, doncs són les que poden portar més problemes o alentir el projecte. A continuació, detallem el raonament per a assignar a cada una aquesta categoria.

En general, les operacions i tasques relacionades amb el render tenen un risc més elevat que la resta. Això és degut al fet que es tracta del resultat visible per a l'usuari, i si no s'obté una qualitat mitjana-alta, la utilitat del projecte i la resta de la feina queda anul·lada. A més a més, tinc menys rodatge treballant amb shaders i GLSL, de manera que l'avanç podria ser més lent.

Tasca 2.5 - Basic Render - Risc: Mig
Tal com s'ha mencionat prèviament, les tasques de render tenen un risc mitjà per defecte.

Tasca 3.1 - Color & Texturing - Risc: Alt
A més a més d'estar relacionada amb el render, una bona distribució de les textures, de manera creïble i interessant, és el que pot fer brillar realment els terrenys. No hi ha molta informació sobre aquest tema, de manera que s'hauran de provar i experimentar amb diferents solucions, el que podria allargar el temps necessari.

Tasca 3.2 LODs - Risc: Mig
Forma part íntegra del terreny, no sols per a optimitzar el render, sinó la generació i la descàrrega. A més a més, tot s'ha de fer de manera dinàmica i tenint en compte la posició del jugador.

Tasca 3.3 LOD stitching - Risc: Alt
Un cop generats els LODs, quedaran forats entre les malles de diferent nivell de detall. Tapar aquests forats trenca la "individualitat" dels chunks, i complica força el codi de generació, el que pot dur a riscos inesperats.

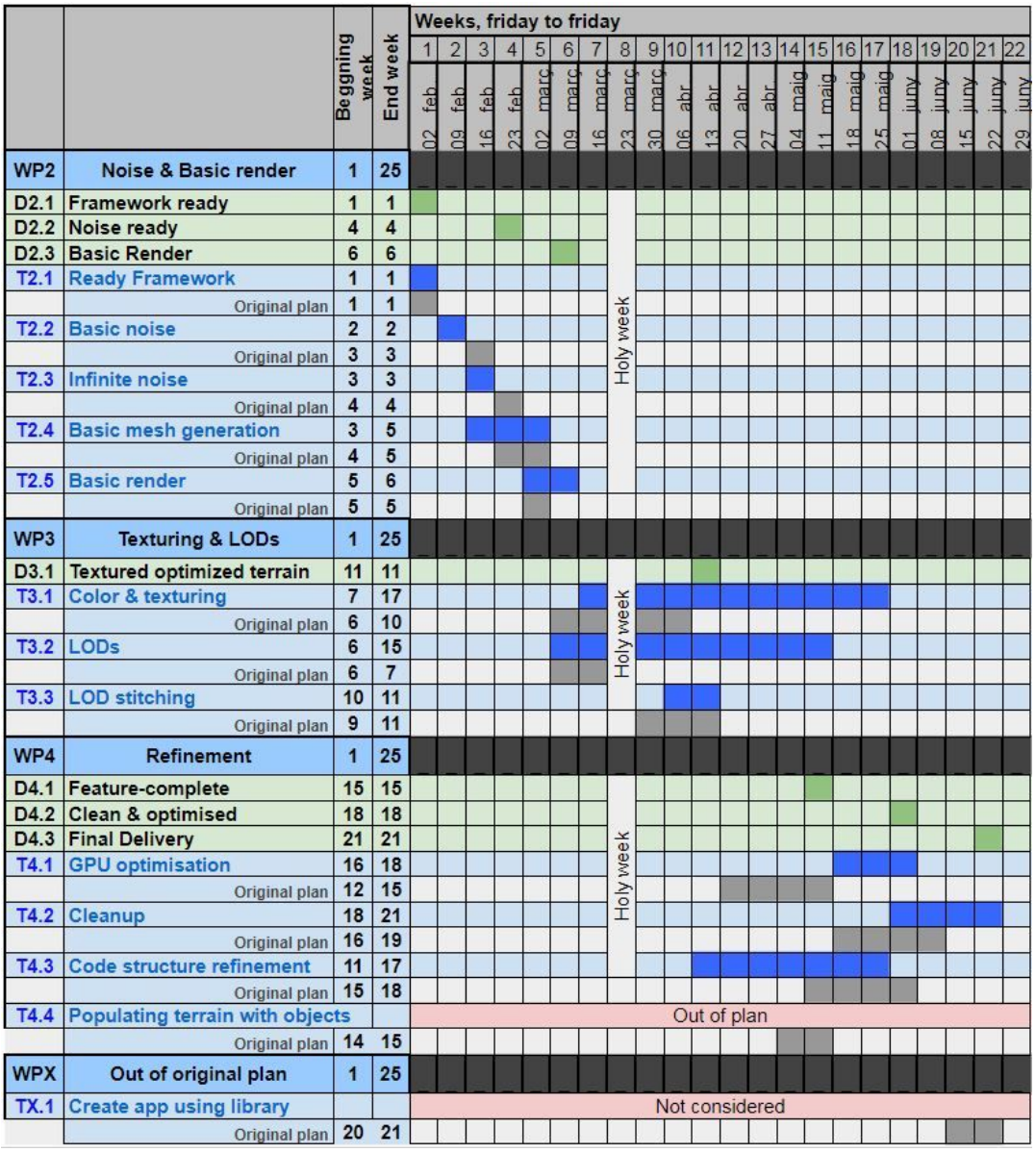
Tasca 4.1 GPU Optimisation - Risc: Molt alt
El desconeixement de què ens podem trobar en aquesta etapa és el risc més gran. Colls d'ampolla, tasques lentes que no es poden estalviar i problemes sense solucionar, tot sense embrutar massa el codi.

Tasca 4.4 Populating terrain with objects - Risc: Alt
El risc d'aquesta tasca no és inherent a la seva dificultat, ja que realment és relativament senzill, sinó amb la complexitat de crear una eina genèrica que permeti poblar qualsevol tipus de terreny amb qualsevol tipus d'objecte que l'usuari pugui definir.

3.5 Desviació de la planificació (27/04/18)

S'han hagut de fer poques modificacions a la planificació original des de l'inici del projecte. Tres canvis majors han ocorregut.

1. La tasca de tessel·lació del terreny (T3.2) ha pres més temps de l'esperat. S'han hagut de provar diferents mètodes i aproximacions, i això ha allargat la tasca més de l'esperat. Encara que això no proposa un retràs al calendari, perquè l'extra de temps dedicat a la tessel·lació i a millorar la seva eficiència, podria formar part de la tasca "GPU optimisation"(T4.1), tasca que hem reduït per a acomodar aquests canvis.
   La feina extra per la tessel·lació és purament d'optimització per a millorar el rendiment del terreny, i poder mostrar més superfície per pantalla.
2. El segon canvi, és l'ampliació de la tasca de Renderitzat de textures (T3.1), substituint la tasca de "Populating terrain with objects" (T4.4). Aquest canvi és degut al fet que, tot i poder tenir textures i colors en el temps estimat, invertir més temps en aquesta tasca ens permetrà obtenir resultats molt més visuals i amb una aparença més polida. En canvi, per a poblar el terreny amb models 3D, i obtenir la mateixa qualitat visual, serien necessaris models 3D de qualitat. A més a més, aquesta "feature" va menys relacionada amb la generació de terrenys en si.
3. La creació de l'aplicació, el joc que utilitzarà com a base la llibreria desenvolupada, corresponent a l'objectiu O.6, no va estar contemplada inicialment a la planificació i s'ha hagut d'afegir posteriorment. Això és degut al fet que el desenvolupament d'aquesta aplicació externa a "C++ RPG Terrains" no serà documentada en aquest document, i serà posterior a l'entrega d'aquesta documentació. Per tant, la seva creació no serà incorporada dins aquest TFG.



		Beggning week	End week
WP2	Noise & Basic render	1	25
D2.1	Framework ready	1	1
D2.2	Noise ready	4	4
D2.3	Basic Render	6	6
T2.1	Ready Framework	1	1
	Original plan	1	1
T2.2	Basic noise	2	2
	Original plan	3	3
T2.3	Infinite noise	3	3
	Original plan	4	4
T2.4	Basic mesh generation	3	5
	Original plan	4	5
T2.5	Basic render	5	6
	Original plan	5	5
WP3	Texturing & LODs	1	25
D3.1	Textured optimized terrain	11	11
T3.1	Color & texturing	7	17
	Original plan	6	10
T3.2	LODs	6	15
	Original plan	6	7
T3.3	LOD stitching	10	11
	Original plan	9	11
WP4	Refinement	1	25
D4.1	Feature-complete	15	15
D4.2	Clean & optimised	18	18
D4.3	Final Delivery	21	21
T4.1	GPU optimisation	16	18
	Original plan	12	15
T4.2	Cleanup	18	21
	Original plan	16	19
T4.3	Code structure refinement	11	17
	Original plan	15	18
T4.4	Populating terrain with objects		
	Original plan	14	15
WPX	Out of original plan	1	25
TX.1	Create app using library		
	Original plan	20	21

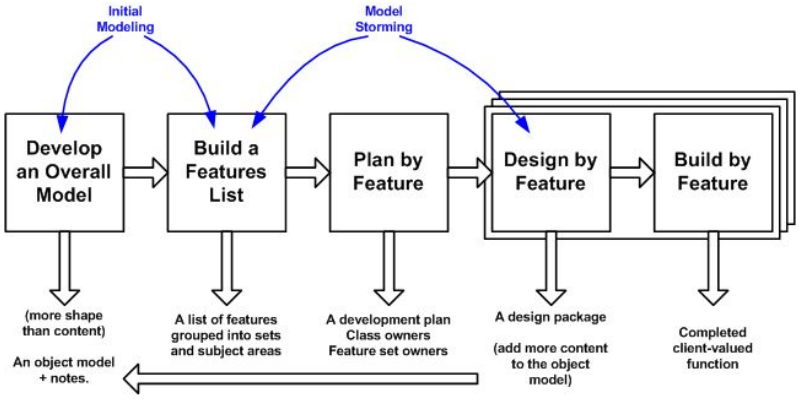
T 3.5 - Desviació del Gantt

4. Metodologia

Per a aquest projecte, s'ha utilitzat una metodologia àgil coneguda com a "**Feature-driven development**" (FDD) per a controlar el progrés realitzat i els passos a seguir en cada moment del desenvolupament.

Aquesta metodologia, com el seu nom indica, es centra en les "Features" que es volen aconseguir, tractant-les com a unitats independents (tot i que s'hi poden establir dependències de bloqueig).

4.1 Feature-Driven Development



F 4.1 - FDD

Aquesta figura, 4.1, representa un esquema dels diferents passos de la metodologia FDD. Les diferents etapes d'aquest model es poden agrupar de diferents maneres però, per a major claredat, les dividim en dos grups.

4.1.1 Model del projecte

En aquesta primera etapa, es treballa sobre el projecte sencer, es planeja, es dissenya. No es comença a construir el software en sí, sinó que es prepara tot el material necessari per a poder-ho fer. Aquest sistema està pensat per a treballar en equips, de manera que el fet de ser un únic integrant simplifica bastant aquestes primeres tasques.

- El primer pas, "Develop an Overall Model", es refereix a clarificar l'objectiu final. Tenir clar on es vol arribar, donar forma al projecte i tenir una visió de totes les necessitats que es volen cobrir així com tots els requisits que s'han de complir.
- El segon pas, "Build a Features List", és desglossar la idea del software acabat en petites tasques o "features", és a dir, cada una de les funcionalitats que l'aplicació necessitarà. Aquesta fase és important, i és necessari fer-la de manera rigorosa, doncs unes features mal definides poden dur a un projecte coix, amb mancances, o feina poc clara o mal ordenada.
- El tercer pas, "Plan by feature", és un pla general. El que busca aquesta etapa és organitzar el llistat de features creat per tal de crear un ordre i establir una prioritat a l'hora d'executar-les. S'han d'establir les dependències (quines tasques han d'estar completades abans de poder començar una tasca) i prioritats.

4.1.2 Feature development

Un cop planejades totes les features i amb un ordre clar en ment, podem començar a desenvolupar. Així com l'etapa anterior sols cal dur-la a terme un cop per projecte, a no ser que s'hi iteri i s'hagin de fer modificacions, aquesta etapa es repeteix per cada una de les features que s'hagin detectat.

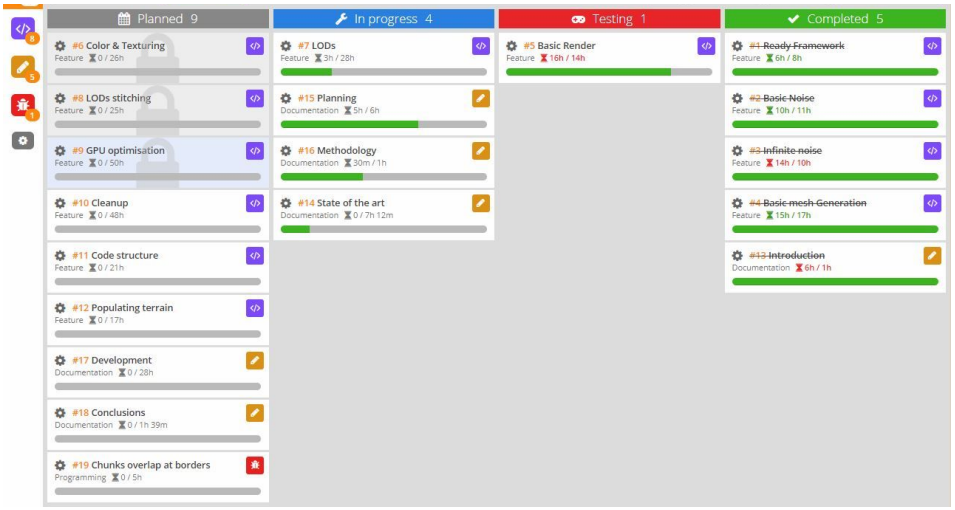
En aquesta etapa es treballa sobre cada una de les "features" individualment, desenvolupant-les i afegint-les al resultat final. Idealment, si la planificació és correcta, un cop totes les features estiguin completes, el resultat serà un producte acabat. Per cada feature, hi ha dos passos, encara que per a aquest projecte hi afegirem un tercer.

- Primer, "Design by Feature". Dissenyar, planejar. Però per a aquesta feature únicament, no pel projecte complet. Establir un full de ruta que cal seguir per a completar aquesta feature en concret.
- Després, "Build by Feature". Com el seu nom indica, es tracta del moment de realment desenvolupar i crear la feature.
- Finalment, "Testing". No podem donar per acabada la feature sense assegurar-nos que funciona. Un cop testejada i donada per bona, podem passar a la següent feature.

4.2 Eines utilitzades per al seguiment del projecte

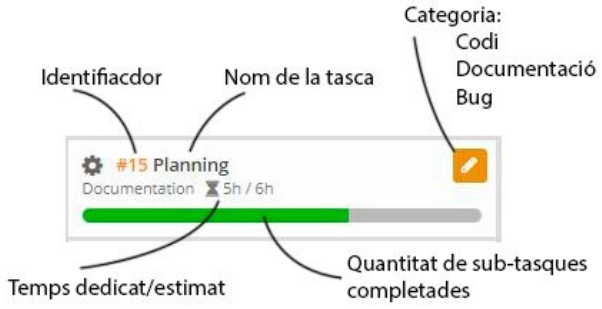
Per a fer un seguiment de l'estat del projecte s'ha utilitzat una eina en línia anomenada "HackNPlan"(16),que permet dur un control de les tasques de manera ràpida, senzilla i eficient, a més a més de cenyir-se a totes les necessitats del FDD.

La figura 4.2 és la interfície de HackNPlan. Podem observar com totes les tasques estan ordenades en quatre columnes segons l'estat en què es troben: "Planificades", "En progrés", "En testeig" i "Completades". Això ens permet identificar de manera ràpida l'estat global del projecte.



F 4.2 - HacknPlan

Cada una de les tasques ens proporciona tota la informació bàsica necessària per poder determinar la feina feta/restant per a aquella feature, així com l'apartat al qual pertany. A més a més, podem observar com algunes d'elles tenen candaus sobreposats. Aquestes tasques estan bloquejades, com hem mencionat anteriorment, doncs algunes tasques poden requerir altres per a poder ser executades, i en HackNPlan podem establir aquestes dependències.



F 4.3 - HacknPlan, Tasca

Una altra de les capacitats més destacades de HackNPlan són les sub-tasques. Cada una de les tasques pot contenir un llistat de sub-tasques que, en estar completades, haurien de completar la feature. La representació gràfica d'aquestes sub-tasques ens permet entendre el progrés que es realitza diàriament.

4.3 Mètode de validació dels resultats obtinguts

4.3.1 Tasques, “Definition of Done”

En completar qualsevol de les tasques, aquesta passava a estar "En testeig". En ser un únic desenvolupador, i en tractar-se d'una llibreria "tècnica", no és fàcil testejar per a errors en una àrea concreta del projecte, pel que es va adoptar una tècnica més genèrica.

Les tasques en testeig es consideren en revisió, i no podran sortir d'aquest estat fins que una nova tasca hi entri, és a dir, fins que s'hagi acabat una altra tasca. D'aquesta manera, sempre hi ha una tasca en desenvolupament i una en revisió, a la que s’ha de parar especial atenció quan executem l'aplicació. Un cop passat aquest període de prova, si no hi ha errors, una tasca es pot considerar “acabada”.

Cal mencionar que, l’aparició de bugs menors no treu la tasca de l’estat de “acabada”, el bug en sí es considera una nova tasca a resoldre.

4.3.2 Validació dels objectius

Hi ha dos tipus d’objectius a l’hora de validar:

- Els subjectius, en cas que el resultat és difícilment quantificable, com ara aconseguir un "Render de qualitat final". Per a aquests, s'haurà de definir si han estat assolits entre jo, el desenvolupador, i el tutor.
- I els quantificables, com obtenir un codi "òptim, amb impacte mínim". Com a hardware de referència per a aquests, utilitzarem els ordinadors de la universitat, amb les especificacions de la taula 4.1.

Processador	Intel(R) Core(™) i7 CPU 2.80GHz
RAM	8.0GB
Sistema	x64
Targeta Gràfica	NVIDIA Quadro FX580

T 4.1 - Especificacions ordinador

4.3.3 Validació dels blocs

A continuació, descriurem el resultat esperat al final de cada bloc, per a comparar-lo amb el resultat obtingut al llarg del desenvolupament.

Bloc 2: Soroll i render bàsic
Al final d'aquest bloc hauríem de ser capaços de visualitzar terreny generat de manera dinàmica al voltant del jugador mentre aquest es desplaça. El terreny no tindrà materials, i els límits seran visibles, doncs sense les optimitzacions necessàries no es podran mantenir molts chunks alhora.

Bloc 3: Texturing i LODs
El terreny ja té textures i colors, i readaptar dinàmicament la qualitat dels chunks ens permet generar un terreny molt més gran sense que afecti el rendiment. Tot i que es podrà mostrar molt més terreny alhora per pantalla, la generació de nous fragments de terreny segueix sent lenta, i pot alentir el funcionament de l'aplicació.

Bloc 4: Refinament
Ja s'hauria de poder visualitzar un terreny virtualment infinit, sense forats. Texturitzat i amb els materials adequats. El codi és net i preparat per a compilar en forma de llibreria, i els Frames Per Segon (FPS) són estables.

Final: TFG
Per a demostrar la utilitat de la llibreria, desenvoluparem un petit joc, un simulador de vol, que l'implementi, per a verificar la seva utilitat dins un possible projecte real.

5. Desenvolupament de C++ RPG Terrains

5.1 L'*engine* base

La llibreria no es pot executar de manera independent, de manera que ha d'estar inserida dins un altre codi. Aquest codi és l'engine, el motor de joc que vaig desenvolupar prèviament a l'assignatura de "Game Engines" a la universitat.

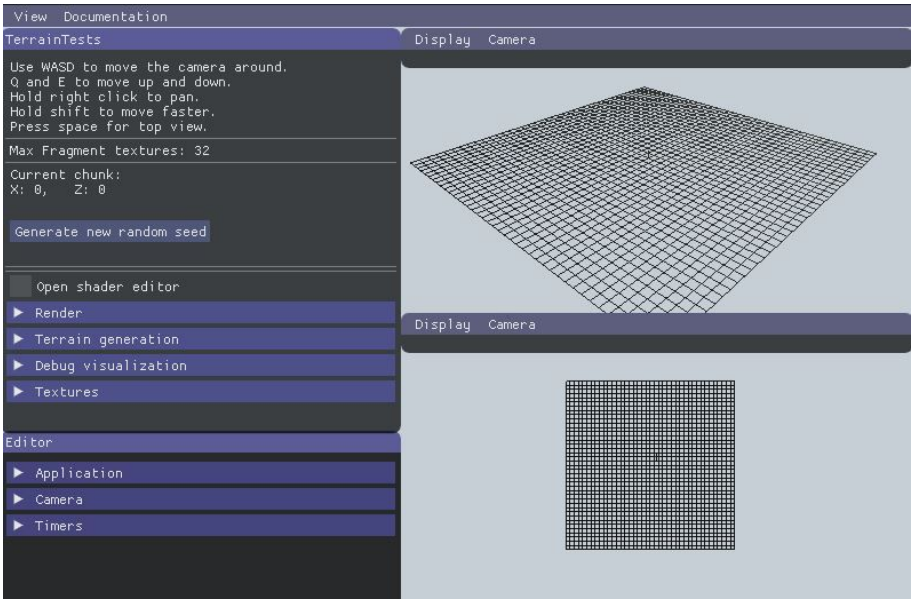
L'engine és "V_Engine" (31), tot i que per a aquest projecte utilitzarem únicament les seves funcionalitats bàsiques. Per això, abans d'iniciar el desenvolupament de la llibreria de terrenys, es va fer neteja del codi per a convertir l'engine únicament en un visualitzador de models 3D, amb funcionalitats bàsiques per a desplaçar-se per l'escena, editar shaders i carregar models i textures.

A més a més, ens permet crear una interfície gràfica per a interaccionar i modificar la configuració de la llibreria de terrenys.

Utilitza diverses llibreries externes:

- Assimp: llibreria per a carregar diversos formats de models 3D.
- Bullet: llibreria de físiques 3D, utilitzada per a detectar col·lisions entre objectes 3D.
- DevIl: llibreria per a carregar imatges 2D.
- Glew: llibreria que facilita la utilització d'OpenGL.
- ImGui: llibreria per a implementar interfícies gràfiques, UI, de manera ràpida i eficient.
- MathGeoLib: llibreria matemàtica, per a treballar amb vectors, matrius i geometria.
- PhysFS: llibreria que permet crear sistemes d'arxius virtuals.
- PuguiXML: llibreria per a llegir i escriure fitxers XML.
- SDL_mixer: llibreria per a carregar i reproduir sons.

A partir d'aquesta base, es desenvolupa la llibreria.



F 5.1 - Captura de V_Engine

5.2 Generació de soroll

Com ja s'ha discutit en el "State of the art", secció 2, la base per a generar terreny procedural és un soroll, que determina les característiques del terreny. A continuació es discuteix el procés de generació del soroll Perlin i les modificacions que s'hi poden realitzar. Per a obtenir resultats més interessants.

5.2.1 Perlin noise

Perlin noise bàsic

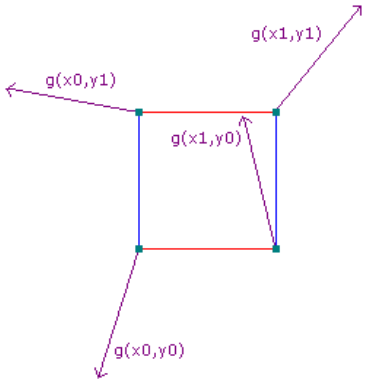
El soroll Perlin permet generar soroll en 2D o 3D. Retorna un valor entre 0 i 1 donada una coordenada X, Y i Z. Una mateixa coordenada sempre retornarà el mateix valor, si ha estat inicialitzat amb la mateixa llavor.

El soroll s'obté a partir d'un array de vectors generat de manera aleatòria, a partir de la llavor donada. D'aquesta manera, generant el soroll a partir d'aquests vectors pre-generats, podem limitar l'aleatorietat i obtenir el mateix resultat donant les mateixes coordenades dos cops. Això també provoca que el soroll no sigui realment infinit, sinó que a partir d'un cert punt, es repetirà en bucle. Això ho podem arreglar de la manera que veurem més endavant.

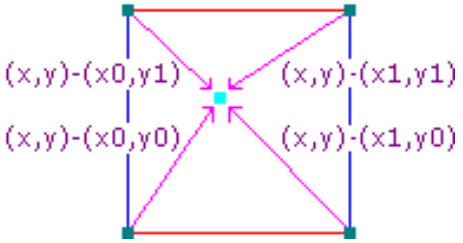
Aquests vectors es distribueixen un una reixa, una grid, de la que s'obtindran els valors de soroll amb el següent càlcul:

A partir de la posició rebuda (x, y), es localitza dins de quina cel·la dins la "grid" cau aquest punt. Així obtenim dos grups de quatre vectors:

- Els quatre vectors corresponents a les cantonades de la casella en què recau la posició, com es pot veure en la figura 5.2.
- Els quatre vectors que representen la distància entre les cantonades i la posició (x, y), com es pot veure en la figura 5.3.



F 5.2 - Perlin generated vectors



F 5.3 - Perlin position vectors

A partir d'aquests vectors, obtindrem quatre valors, el producte escalar corresponent entre cada un dels vectors cantonada i distància obtinguts prèviament (cada un dels vectors cantonada amb cada una de les distàncies a aquella cantonada).

A continuació, fem una interpolació linear entre aquests quatre valors, a partir de la posició x i y dins la cel·la. El valor resultant és el valor del soroll Perlin en la coordenada (X, Y).

Perlin noise fractal

El mètode anterior és la base de tot el soroll perlin, però produeix un resultat sense interès i poc útil, especialment per a utilitzar de base per a terrenys. És per això que s'introdueix el concepte dels fractals, per a crear variació i detall.

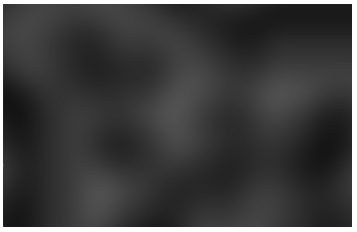
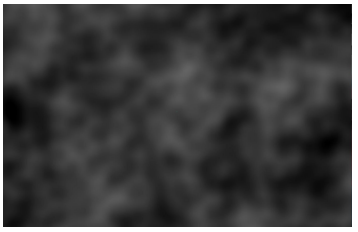
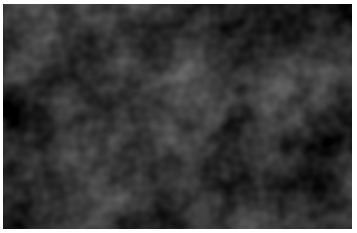
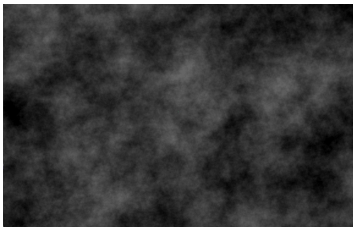
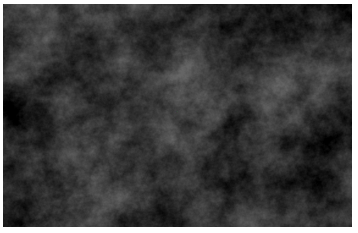
El concepte és senzill: generem diferents mapes de soroll, de diferents mides i els sumem, amb diferents intensitats. Podem veure el funcionament a la figura 5.4, que mostra la implementació dels fractals a la generació de soroll del projecte. La funció "noise(x,y,z)" retorna el resultat del perlin noise per a aquella coordenada.

```
float octaveNoise(float x, float y, float z, std::int32_t octaves, float lacunarity = 2.f, float persistance = 0.25f)
{
    float result = 0.0f;
    float amp = 1.0f;

    for (std::int32_t i = 0; i < octaves; ++i)
    {
        result += noise(x, y, z) * amp;
        x *= lacunarity;
        y *= lacunarity;
        z *= lacunarity;
        amp *= persistance;
    }

    return result;
}
```

F 5.4 - Codi Soroll Fractal

Per a això, introduïm tres noves variables: el nombre d'octaves, la llacunaritat i la persistència.

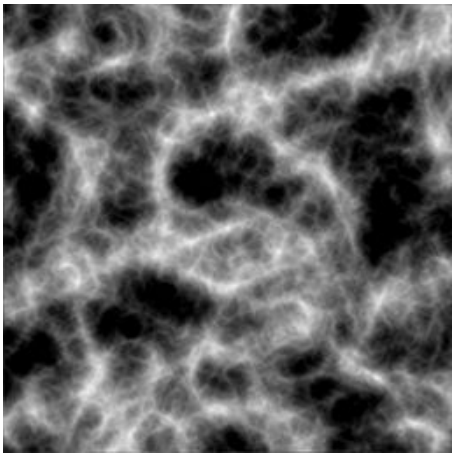
- Llacunaritat: multiplicador aplicat a la coordenada (x,y,z) per cada nou nivell generat, creant així mapes amb una freqüència major o menor segons el seu valor.
- Persistència: reducció aplicada a la influència d'aquell nivell, que es redueix a cada nova octava generada.
- Nombre d'octaves: quantitat de mapes de soroll generats. Una major quantitat oferirà més detall, però també serà més lent. Podem visualitzar l'efecte de diferents quantitats d'octaves a la taula 5.1.

 1 octava	 2 octaves	 3 octaves
 4 octaves	 8 octaves	 16 octaves

T 5.1 - Perlin noise octaves

Ridged perlin noise

Una nova variació del Perlin noise, és el "Ridged Perlin Noise", o soroll perlin amb "crestes". Es tracta d'una petita modificació al soroll perlin normal, que permet crear aquest efecte de "crestes", un exemple visible a la figura 5.5. Això és ideal per a similars serralades i un efecte més natural i realista que el soroll perlin sense modificacions.



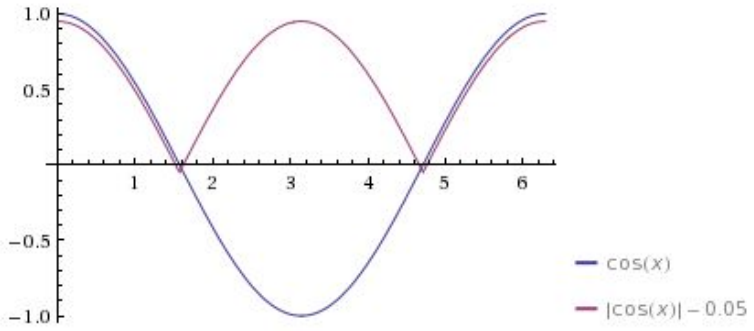
F 5.5 - Ridged perlin noise

El podem obtenir simplement aplicant la següent fórmula al resultat del soroll:

r =1-abs(p * 2 - 1)

Sent "p" el resultat del soroll perlin previ, en el rang entre 0 i 1, i "r" el resultat que ens proporciona la variació "ridged".

El raonament rere aquesta fórmula és sorprenentment senzill. Comprovem l'efecte que té aplicar "abs(cos(x))", a la figura 5.6.



F 5.6 - abs(cos(x))

Podem observar com aplicar el valor absolut (en vermell) sobre una funció cosinus (en blau) crea pics a la funció. A més a més, en crear els "pics" en el valor central en comptes d'en els màxims i mínims, farà que siguin més freqüents. Aquests pics són els que es convertiran en les crestes del "ridged perlin noise". Invertint el resultat d'aquesta funció, "1 - abs(cos(x))", els pics queden a la part superior, un resultat més idoni per a crear muntanyes.

Tot i això, aquest raonament únicament funciona en rangs de valors de -1 a 1. És per això que apliquem "p * 2 - 1" sobre el valor del soroll perlin, perquè encaixi entre aquests valors en comptes d'entre 0 i 1.

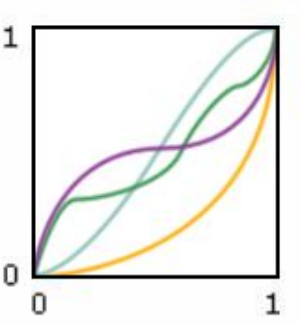
Dins aquest treball, al treballar amb octaves, tenim l'opció d'aplicar el soroll ridged a partir d'una octava concreta, per a una major varietat de resultats.

Corba d'alçades (height curve)

Una altra modificació que apliquem al terreny és la height curve. Es tracta d'una funció senzilla que rep un valor de 0 a 1 i retorna un altre valor de 0 a 1. Alguns exemples representats gràficament de height curve a la dreta, figura 5.7.

Aquesta funció s'aplicarà a cada un dels píxels del mapa de soroll, modificant l'aparença global del terreny.

Això ens permet ajustar la topografia del terreny, per a generar platges, platees o pics escarpats, oceans, illes o el que es desitgi.



F 5.7 - Height curves

Enviromental maps

Una altra eina utilitzada per a afegir variació al terreny són els "enviromental maps". Aquests mapes, consistents també d'un soroll perlin, determinen diferents factors ambientals a partir dels quals es poden ajustar les textures o les propietats del terreny.

És habitual crear mapes d'humitat i de temperatura que, conjuntament amb l'alçada del terreny, permeten diferenciar diferents "biomes" o regions dins el terreny.

Cada una d'aquestes regions és continguda dins un rang de valors concrets, i pot modificar l'aparença del terreny en general.

5.2.2 Soroll Infinit

El soroll perlin, per defecte, no te final i és capaç de generar valors per qualsevol coordenada (x,y,z). Tot i que el soroll sense fractals, un cop arribat al final de l'array de vectors sí que repetirà el mateix mapa soroll altre cop en bucle, l'adició de fractals fa que aquesta limitació sigui pràcticament imperceptible, doncs fins i tot després de la repetició els diferents fractals aporten suficient variació per a què el terreny no sigui reconeixible, i la repetició real ocorri en distancies suficientment grans com perquè no sigui un problema real.

5.2.3 Optimitzacions

La generació del soroll pot ser lenta, segons la resolució del mapa de soroll que generem i la quantitat d'octaves que es generin. És per això que és necessari aplicar algunes optimitzacions per a evitar l'alentiment del programa i poder generar qualsevol mapa de soroll a temps real.

Generació en un fil d'execució paral·lel

La primera optimització, si bé no accelera la generació, evita l'alentiment de la resta del programa i minimitza el risc què baixin les imatges per segon.

Separem la generació del soroll a un fil diferent del principal utilitzant una classe senzilla anomenada "Chunk Factory", que conté principalment els mètodes mostrats a la figura 5.8. Utilitzant "std::thread" i "std::mutex", podem evitar els problemes d'asincronisme.

```
class ChunkFactory
{
public:
    void LaunchThread();
    void StopThread();

    void SetSeed(uint seed);

    void PushChunkRequest(RequestedChunk request);
    GeneratedChunk PopGeneratedChunk();
    bool HasGeneratedChunks();
```

F 5.8 - Codi Chunk Factory

Guardar les normals en el mapa de soroll

Per a evitar més càlculs al fil principal, també calculem les normals dels vèrtexs generats dins la "ChunkFactory" i les guardem dins el noise map. En tractar-se d'una textura de color, ens permet aprofitar els quatre canals, RGBA, guardant els tres components del vector normal i el valor del soroll.

Per a calcular la normal d'un vèrtex, fem el producte escalar entre els vectors que formen els quatre costats que contenen aquell vèrtex i fem la mitjana entre ells.

5.3 Generar meshes

A l'engine, dibuixem a través d'OpenGL utilitzant els vèrtexs de la mesh i els índex, usant "glDrawElements()", de manera que s'han de generar tots dos per separat.

Els índexs han estat sempre estàtics, compartits per totes les meshes del terreny. Ja que tots els chunks tenen les mateixes dimensions i els vèrtexs organitzats de la mateixa manera, podem reaprofitar el buffer d'índex per a totes les meshes del projecte, necessitant així un únic buffer amb els índexs. Comentarem amb més detall la seva generació a l'apartat "5.4.3 - LODs", ja que han estat molt vinculats amb els nivells de detall.

El procés de generació de meshes ha passat per dues etapes diferenciades durant el desenvolupament del projecte. Inicialment, per a poder començar a testejar i previsualitzar els resultats de generació de soroll, es generaven directament les meshes finals que serien renderitzades, una mesh diferenciada per a cada chunk. Actualment, existeix una única malla que és instanciada múltiples cops i editada a través dels shaders per a mostrar-la com sigui necessari.

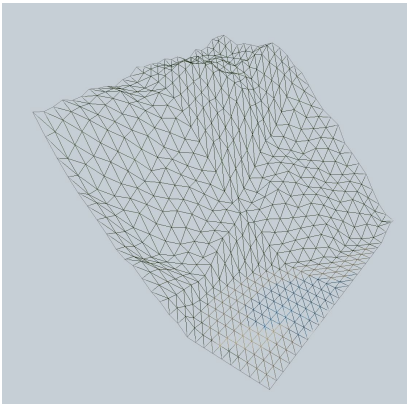
Les diferents etapes per què ha passat la generació de meshes és:

- **Generar el resultat final**

Inicialment, es generava directament el resultat final que es renderitzaria sense haver d'aplicar cap transformació ni modificar la mesh a través de shaders. En carregar un nou chunk es generava una mesh única per a ell i, en generar cada vèrtex, s'obtenia el valor del soroll per a aquella posició i s'aplicava directament a la y del vèrtex, multiplicat per l'alçada màxima designada a la configuració.

L'avantatge d'aquest mètode és la seva senzillesa: després de crear la mesh no cal utilitzar shaders per a realitzar cap modificació sobre el resultat. Desafortunadament, això no permet cap mena de personalització, per a fer qualsevol modificació a l'alçada o la mida dels chunks, és necessari regenerar totes les meshes existents.

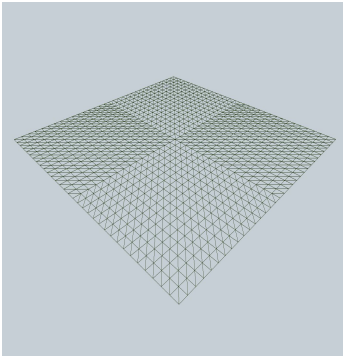
Els chunks tenien una resolució de chunk.size per chunk.size, podent modificar la resolució a partir de la configuració, però necessitant regenerar totes les meshes perquè tingui efecte.



F 5.9 - Generació de meshes 1

- **Generar les alçades entre 0 i 1**

El segon pas, era generar les meshes amb alçades entre 0 i 1 i multiplicar el valor per l'alçada màxima del terreny en el vèrtex shader. D'aquesta manera, es pot modificar l'alçada sense haver de tornar a crear les meshes. Cada chunk encara ha de tenir la seva mesh única, ja que és la que emmagatzema la informació necessària.



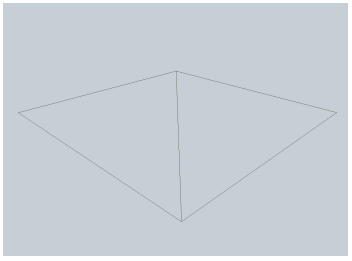
F 5.10 - Generació de meshes 2

- **Generar superfícies planes de chunk.size**

Arribats a aquest punt, generava una única mesh plana, amb resolució de chunk.size per chunk.size. Les alçades són enviades al shader, enviant directament el mapa de soroll com a textura. D'aquesta manera, instanciant una única mesh i enviant el mapa de soroll corresponent podem dibuixar tots els chunks, sense necessitar més d'una mesh per a tot el terreny.

- **Generar un quadrat de 1x1**

Un cop la llibreria suportava tesselation, la mesh es va poder simplificar encara més: l'única malla necessària era un pla d'1 per 1, amb dos triangles, amb mida d'una unitat. La transformació i les alçades són aplicades ja totes a través dels shaders.



F 5.11 - Generació de meshes 3

5.4 Render

5.4.1 Texturitzat

Textures condicionals

Per a poder texturitzar el terreny, no podem definir manualment on anirà cada textura, de manera que optem per a utilitzar "textures condicionals": textures amb condicions que determinen en quines condicions es col·locaran sobre el terreny. Podem veure a la figura 5.12 el recull de condicions aplicades a cada una de les textures.

```
    Vec3<float> color = Vec3<float>(1.f,1.f,1.f);
    float minSlope = 0.f;
    float maxSlope = 1.f;
    float minHeight = 0.f;
    float maxHeight = 0.f;
    float sizeMultiplier = 4.f;
    float heightFade = 10.f;
    float slopeFade = 0.01f;
private:
    float hasTexture = 0.f;
```

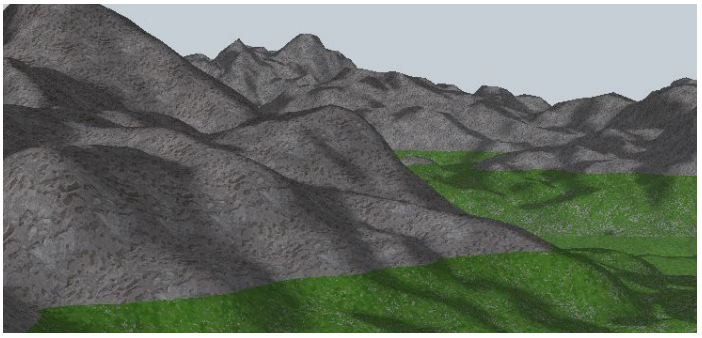
F 5.12 - Codi Conditional texture

D'aquesta manera, podem distribuir les textures segons l'alçada en què es trobi cada vèrtex i el pendent. Així podem aconseguir efectes com, per exemple, tenir neu a partir de certa alçada, però únicament a les zones amb poc pendent, i roca a les zones on no hi hagi neu.

Texture displacement

Aquesta tècnica funciona sobre el paper, però el resultat pot ser simplista o li pot faltar complexitat. En certs punts del terreny, el canvi entre una textura i una altra es pot convertir en una línia horitzontal en arribar a l'alçada designada com a màxim o mínim. Això resta credibilitat al texturitzat, i fa que sigui menys natural. Veure exemple a la figura 5.13, texture height condition.

Això es pot arreglar generant un heightmap secundaris, per a designar el "displacement" dels valors de la textura. Aquests valors se sumaran als valors de la textura condicional, per a aplicar-hi una certa variabilitat, augmentant o disminuint les alçades màximes i mínimes, per a eliminar l'efecte de "tall" entre dues textures contigües que poden coexistir en les mateixes normals.



F 5.13 - Límit d'alçada de textures

Texture blending

Per a millorar l'aparença de les textures, necessitem alguna mena de "blend" entre elles, més enllà d'una línia sòlida entre totes dues.

És per això que, com es pot veure a la figura 5.12, Conditional texture in code, tenim també les variables de "heightFade" i "slopeFade". Aquests valors ofereixen la possibilitat de designar una zona de "fade" entre dues textures adjacents. Per exemple, si una textura té com a "maxHeight" 100 i com a "heightFade" 10, aquesta textura serà sòlida fins a l'alçada 100, i desapareixerà progressivament fins a l'alçada 110.

F 5.14 - Blend lineal de textures

La primera aproximació al fade entre textures, és una mescla lineal entre elles, com podem veure a la figura 5.14 - Blending de textures. El valor d'opacitat de la textura és 100% a "maxHeight" i és 0 a "maxHeight + heightFade", fent una interpolació lineal entre tots dos. De la mateixa manera, el mateix fade s'aplica per a les normals i l'alçada mínima.

Tot i això, aquest resultat és poc realista i, mentre és prou vistós, no és el millor que podem aconseguir visualment.

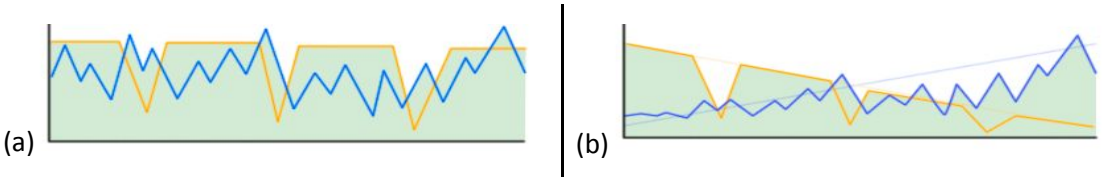
Heightmap blending

Per a millorar el resultat del "blending", utilitzem la informació proporcionada pel heightmap de la textura per a fer la mescla entre textures.

Cada punt de la textura, té un valor d'alçada, com és representat a la figura 5.15 - Texure heightmap values. Aquest gràfic representa el valor dels heightmaps de dues textures 2D, en una fila de píxels de la imatge. En aquest nou mètode de blending, en comptes de fer una interpolació linear i canviar l'opacitat, pintem la textura que tingui l'alçada major en cada punt, per a oferir una mescla més natural entre totes dues.

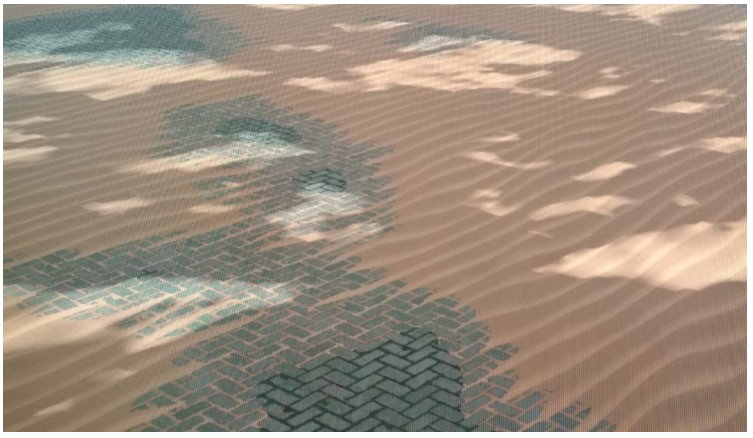
I, per a tenir una mescla progressiva entre totes dues textures, multiplicarem el valor d'alçada de la textura en cada punt pel valor de la interpolació linear de l'opacitat. D'aquesta manera, en comptes de modificar l'opacitat, modifiquem l'alçada màxima de cada un dels punts de la textura, com es pot visualitzar a la figura 5.15, Texture heightmap blending.

Les línies opaques representen el valor del heightmap de cada una de les textures, i les línies més clares són el valor de la interpolació de l'opacitat entre totes dues. En multiplicar aquests valors podem veure com el blend entre textures obté una mescla més gradual que sense.



F 5.15 Texture heightmap blending, on es pot visualitzar el valor dels heightmaps de les dues textures sense modificar (a) i aplicant l'interpolació lineal entre tots dos (b).

A la figura 5.16, Texture blending resultat, podem veure el resultat de mesclar dues textures, de sorra i de maons, utilitzant aquest mètode de blend entre textures.



F 5.16 - Texture blending resultat

Textures 3D

Tot i que per falta de temps no s'ha pogut implementar, una tècnica habitual per a enviar les textures dels terrenys és usar textures 3D.

D'aquesta manera, en comptes d'utilitzar el mètode tradicional d'emmagatzemar una textura en cada buffer i assignar a cada buffer un uniform a través del qual s'hi pugui accedir en els shaders, s'envia una única textura 3D, en què cada capa, cada valor de "z", forma part d'una textura diferent.

D'aquesta manera, es poden accedir a grups de textures, "apilades" unes sobre altres, accedint a un sol buffer.

Alguns dels inconvenients d'aquest mètode són l'increment de complexitat i quan diferents textures tenen diferents mides, perquè l'amplada i llargada de la textura 3D ha de ser la de la textura més gran de cada una de les "layers". Això provoca espais sense utilitzar entre textures, espai que s'està desaprofitant.

5.4.2 LODs

Una de les optimitzacions més importants i necessàries per a renderitzar terrenys és tenir diferents nivells de detall o "LOD"s(Levels of Detail). Això permet modificar la densitat de la geometria segons la distància a la càmera o altres factors, per tal de no renderitzar més triangles dels necessaris i alleugerir la càrrega de la GPU.

Sets d'índexs

La primera aproximació és un mètode que s'utilitzava fa temps en els jocs més antics. A l'hora de generar índex per a dibuixar la mesh, es generen diferents "sets" d'índexs, que dibuixaran més o menys vèrtex segons la densitat desitjada. Després, al moment de dibuixar, es calcula la distància amb el chunk específic i se li assigna el set d'índex adequat segons el nivell de detall a utlitzar.

Per exemple, veure la figura 5.17, Index LODs. A l'esquerra, la mesh completa, la seva aparença quan és dibuixada amb tots els índexs. Al centre, el segon set d'índex, en què es duplica la mida de cada polígon de la malla. I, a la dreta, una tercera iteració en què es disminueix encara més la quantitat de polígons modificant únicament els índexs enviats.

F 5.17 - Index LODs

Tesselation

El segon pas és deixar de modificar els índexs i utilitzar el shader de tessel·lació, introduït amb OpenGL4.0. Aquest shader ens permet dividir la geometria existent i genera nous vèrtexs, de manera similar al geometry shader, però és capaç de treballar de manera més eficient.

Amb aquesta aproximació, necessitem una mesh simplificada a la que puguem afegir detall utilitzant la tessel·lació, contràriament al mètode anterior en què necessitàvem una mesh complexa que poguéssim simplificar.

Un dels paràmetres necessaris per a aquest shader és el "tesselation level", que determina la quantitat de divisions que aplicarà a cada triangle de la mesh. Hi ha diferents mètodes per a obtenir aquest valor.

- **Distance based**

Podem utilitzar la distància a la càmera per a definir un nivell de tessel·lació: com més proper a la càmera es trobi un chunk, més subdivisions se li aplicaran. És una aproximació senzilla i ràpida, tot i que no és la més òptima.

- **Screen size based**

També podem utilitzar la mida per pantalla de cada triangle, per a definir la quantitat de subdivisons, per intentar que tots els triangles mantinguin una mida constant a la pantalla, sense importar la distància.

Això s'aconsegueix aproximant la mida mitjana dels triangles d'un chunk en una esfera (únicament per què les esferes són més ràpides i eficients) i projectar aquesta a la pantalla. A partir dels valors obtinguts, podem deduir la quantitat de subdivisions necessàries per a mantenir la mida dels triangles més o menys constant.

A més a més, aquest mètode ens permet tenir de manera ràpida un "frustum culling" senzill, per a descartar els polígons que es trobin fora de la pantalla i que no siguin pintats. En projectar la geometria a la pantalla per a obtenir la seva mida, podem directament descartar els triangles que es trobin fora de la zona visible, evitant així la seva tessel·lació i posterior crida i "discard" dins el fragment shader.

Tessellation spacing modes

A l'hora de tessel·lar, existeixen també diferents tipus de "spacing", la manera en què es decideix la posició de cada un dels vèrtexs generats dins la geometria existent, i la quantitat de subdivisions a aplicar. Existeixen tres modes de subdivisió, que s'indiquen a l'inici del shader de control (TCS):

- layout(triangles, **equal_spacing**) in;

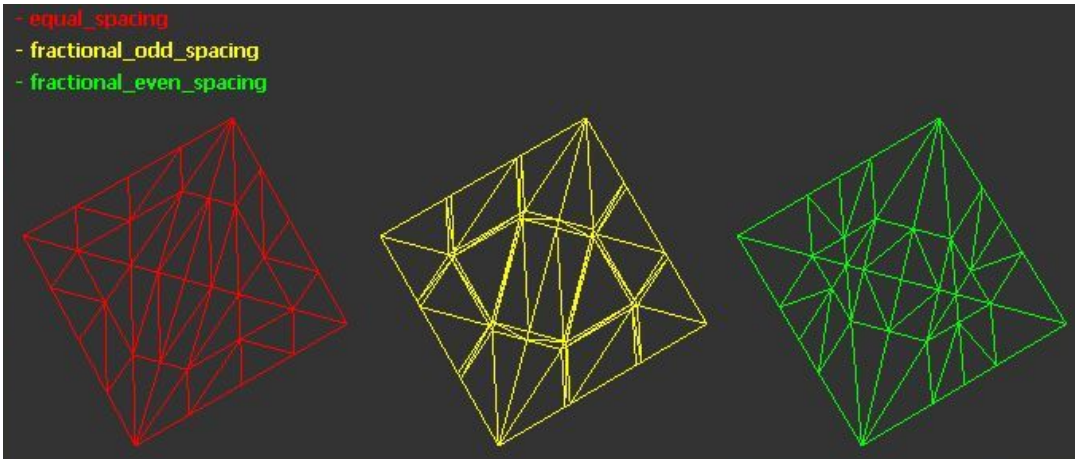
Accepta valors de subdivisió enters entre [1, 64].
La quantitat de subdivisions és igual al valor proporcionat. En cas que sigui un flotant, s'arrodonirà a l'alça.
La distància entre vèrtexs és la mateixa al llarg de tota la vora. Aquest mètode és l'usat finalment en l'aplicació, i el seu resultat es pot veure a la figura 5.21, Resultat de la Tessel·lació.

- layout(triangles, **fractional_odd_spacing**) in;

Accepta valors flotants entre [1, 63].
La quantitat de subdivisions és el valor proporcionat arrodonit a l'alça al nombre senar més proper.
En rebre valors senars enters, la distància entre vèrtexs és igual entre tots. En rebre qualsevol altre valor, les distàncies entre els vèrtexs del valor senar previ són iguals entre elles. Les dues noves subdivisions afegides són més petites que la resta, i tindran la mateixa mida al següent valor senar.

- layout(triangles, **fractional_even_spacing**) in;

Accepta valors flotants entre [1,63].
La quantitat de subdivisions és el valor proporcionat arrodonit a l'alça al nombre parell més proper.
El funcionament és similar al "fractional odd spacing", tot i que treballant amb nombres parells.

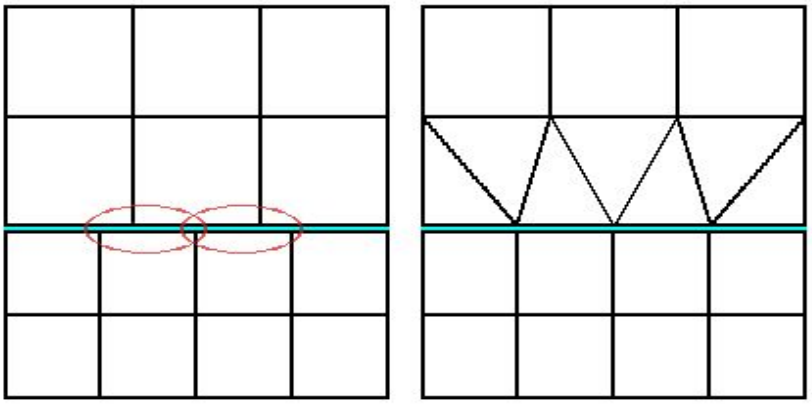


F 5.18 - Tesselation spacing modes

LOD stitching

Tot i la utilitat i la potència de la tessel·lació, pot comportar també una sèrie de problemes que s'han de resoldre, i aquest és el primer. Quan el nivell de subdivisió de dos chunks no coincideix, poden aparèixer artefactes visuals, com els de la figura 5.20, LOD Holes. Això és degut a que la quantitat de polígons del vèrtex d'un chunk i del consecutiu no coincideixen, i la malla resultant no és uniforme i pot tenir forats, com es representa a la figura 5.19, LOD stitching esquema, a). L'alçada de cada un dels vèrtexs no coincidents pot variar, creant els "forats" a la malla.

S'ha hagut de trobar un mètode per a poder obtenir un resultat similar a la mateixa figura, part b), en què s'ha de modificar alguna de les vores perquè coincideixi amb les diferents resolucions dels chunks adjacents.



a) b)

F 5.19 - LOD stitching esquema. A l'esquerra, a), la representació de la malla de dos chunks col·lindants de diferent resolució. A la dreta, b), possible solució per a evitar els forats.

Existeixen diverses solucions, amb diferents aproximacions:

1. **Pre-calcular LOD per chunks**

Es pot calcular la quantitat de subdivisions en la CPU, de manera prèvia al render. D'aquesta manera, es pot enviar al shader la informació de la quantitat de subdivisions que ha de tenir un chunk concret, així com el nivell de subdivisions de tots els chunk veïns.

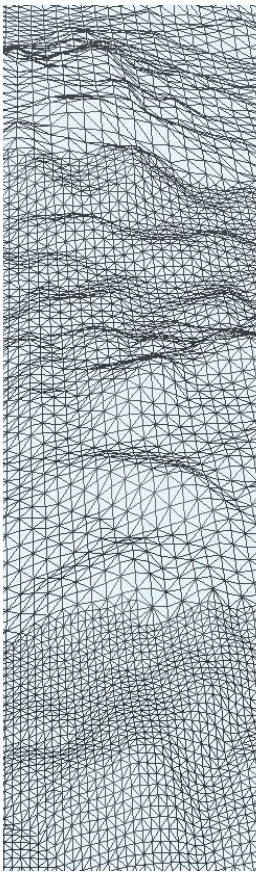
Al shader, amb aquesta informació, es poden modificar tots els chunks que tinguin un veí de menor resolució, per a augmentar la quantitat de subdivisions a la vora necessària, coneixent la quantitat de polígons que té el nostre chunk i la que té el veí.

2. **Ocultar els forats**

Una altra opció és mantenir els forats, però ocultar-los de diferents maneres. Aquesta és una opció vàlida doncs, els forats més grans es trobaran a grans distàncies, i els forats propers seran tots petits, doncs la diferència entre la mida de polígons és molt menor. Podem veure aquesta diferència de mides dels forats a la figura 5.20, LOD Holes.

Això es pot fer col·locant "parets" per la part inferior de cada chunk, o modificant el color del skybox per la part inferior del terreny.

F 5.20 - Forats de LOD



F 5.21 - Resultat de la Tessel·lació

2. **Calcular el LOD per cada vora**

Aquesta és l'opció aplicada a aquest projecte. I és que, en comptes de calcular el LOD segons la distància del chunk o la mida d'aquest a la pantalla, ho fem calculant-l'ho sobre cada una de les quatre vores del chunk. D'aquesta manera, el resultat de l'operació ha de ser el mateix per a tots dos chunks, doncs la mida total de la vora entre chunks no varia, i la distància és la mateixa, doncs són adjacents.

Podem decidir el LOD del chunk com a una mitjana de les quatre vores però, tenint el LOD que ha de tenir cada una de les vores, ens podem assegurar de què cada una d'elles encaixi amb el chunk contigu.

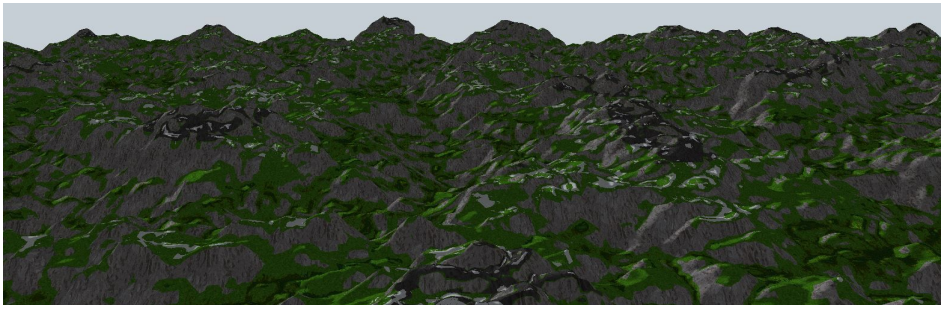
5.4.3 Boira

Un efecte força indispensable es la boira. Ens permet enmascarar els petits errors que puguin aparéixer, i ocultar completament els límits del terreny generat, per a donar la sensació de que el terrent és, efectivament, infinit. A més a més, augmenta la qualitat visual, doncs com podem veure a la figura 5.22 a), totes les montanyes es confonen entre elles i la sensació de distància queda esmorteida per la similitud de color de tot el terreny, convertint la vista en una mescla de colors poc clara.

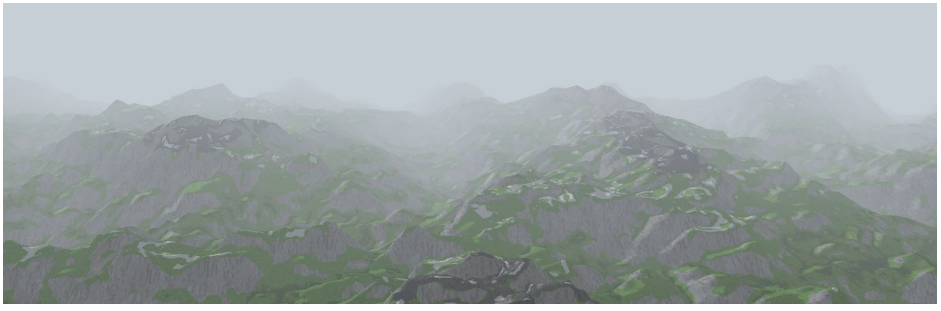
La primera iteració de boira la podem veure a la figure 5.22 b), basada únicament en la distància. Hi ha una distància màxima visible, definida per l'usuari, en que l'opacitat de la boira és del 100%. Des d'allà, es fa un fade segons la distància de cada punt a l'observador. Aquest métode ens permet enmascarar els límits del terreny, a més a més de diferenciar les montanyes properes de les més distants, el que augmenta la sensació de distància i, per tant, fa que tot sembli més gran i més real, palpable.

La segona iteració és visible a la figura 5.22 c), en que també es te en compte l'alçada de cada punt a l'hora de calcular la densitat de la boira, oferint un modificador de fins a *1.5 d'opacitat en l'alçada mínima (en un rang [0, 1]). Aquest simple canvi d'augmentar l'opacitat en les alçades més baixes fa que s'assimili més a la realitat, en que la boira acostuma a ser menys dense en alçades superiors, i crea un efecte "d'halo" entre una montaña i la següent, remarcant la seva silueta i oferint un resultat més interessant.

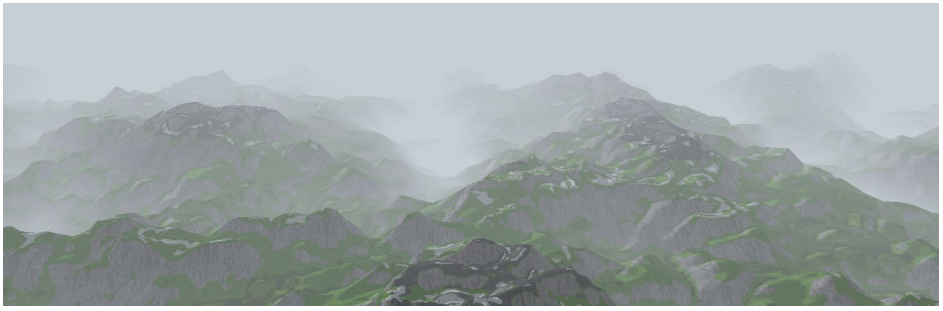
a)



b)



c)



F 5.22 - Efectes de boira. a) Sense boira, b) Boira per distància, c) Boira per distància i alçada

```
float distanceFog = (gl_FragCoord.z / gl_FragCoord.w) / fog_distance;
distanceFog = min(distanceFog, 1);

float heightFog = (1 + water_height - height);
heightFog = pow(heightFog, 3);

float fog = distanceFog + (heightFog * 0.5f) * pow(distanceFog, 0.25f);
fog = min(fog,1);

color = vec4(mix(col, fog_color, fog), 1.f);
```

F 5.23 - Codi boira

5.5 Chunk Management

Fins ara hem estat parlant únicament dels chunks i del seu funcionament, però necessitem també un objecte que s'encarregui de coordinar la creació i destrucció d'aquests chunks. Es tracta d'una estructura senzilla que conté la fàbrica de chunks i treballa conjuntament amb ella per a generar i emmagatzemar els chunks.

La seva base és senzilla, i el funcionament no té complexitat. La seva funció més important, l'update, es pot veure a la figura 5.24, Codi Chunk Manager. Si el jugador s'ha desplaçat a un chunk diferent, s'afegeixen els nous chunks a la fàbrica per a ser generats i si en qualsevol frame la fàbrica té nous chunks generats, substitueix els chunks sense carregar o els més llunyans pels nous chunks.

```
while (m_factory.HasGeneratedChunks())
{
    GetFurthestChunk().Regenerate(m_factory.PopGeneratedChunk());
}

if (currentPosition != lastPos)
{
    m_factory.ClearRequests();
    lastPos = currentPosition;
    AddChunksToRegen(currentPosition);
}
```

F 5.24 - Codi Chunk Manager

5.6 Interacció amb la llibreria

Totes aquestes funcionalitats han de quedar el més aïllades possibles de l'aplicació externa que utilitzi la llibreria, per a simplificar el seu ús i per a oferir una interfície d'ús clara i intuïtiva que qualsevol desenvolupador pugui utilitzar per a treballar amb ella.

Aquesta interacció està dividida en dos fitxers, amb diferents objectius:

```
struct Config
{
    unsigned int maxChunks = 512u;
    float chunkSize = 1024.f;
    unsigned int chunkHeightmapResolution = 32u;
    unsigned int chunkMinDensity = 10;

    float maxHeight = 1000.f;
    float waterHeight = 0.004f;

    float fogDistance = 10000.f;
    float fogColor[3] = { 0.78f, 0.81f, 0.84f };

    float ambientLight = 0.4f;
    float globalLight[3] = { 0.2f, -0.2f, 0.2f };
    bool singleSidedFaces = true;

    struct Noise
    {
        unsigned int ridgedDepth = 2;
        float frequency = 0.404f;
        unsigned int octaves = 5u;
        float lacunarity = 2.0f;
        float persistency = 0.428f;
    } noise;

    struct Debug
    {
        bool wiredRender = false;
        bool renderLight = true;
        bool renderHeightmap = false;
        bool renderChunkBorders = false;
    } debug;

    std::function<void(const char*)> throwErrorFunc;
};
```

Fitxer "Config.h", una captura del qual es pot veure a la figura 5.25, Arxiu "config.h". Aquesta estructura conté dades que haurien de ser setejades a l'inici de l'execució i no s'haurien de modificar al llarg de l'execució. Exceptuant algunes que afecten únicament al render, la majoria influeixen la generació del terreny.

Modificar aquestes variables en temps d'execució farà que els nous chunks generats no encaixin amb els existents. És per això que serà necessari esborrar els chunks existents o regenerar la mesh, després de modificar aquestes opcions, segons les que s'hagin modificat.

F 5.25 - Arxiu “config.h”

F 5.26 -Arxiu “include.h”

```
void Init();
void Update(int posX, int posY);
void Render(const float * viewMatrix, const float * projectionMatrix);
void SetHeightCurve(std::function<float(float)> func);
void CleanChunks();
void RegenerateMesh();
void SetSeed(unsigned int seed);

std::string CompileShaders(const char * frag, const char * vert, const char* TCS, const char* TES);

float GetHeightAt(float x, float y);

const ConditionalTexture& GetTexture(int n);
void SetTexture(int n, const ConditionalTexture& tex);
```

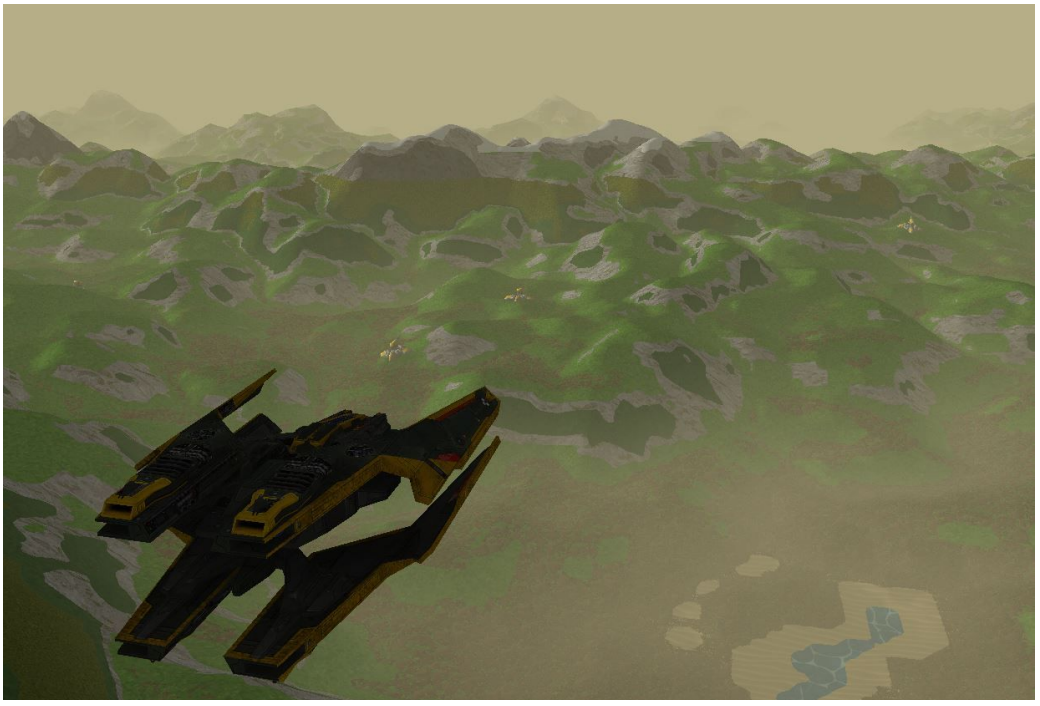
A la figura 5.26, Arxiu "include.h", podem veure les funcions globals definides al fitxer "Include.h". Aquestes són totes les funcions a les quals té accés l'usuari extern, i amb les quals es pot fer funcionar tota la llibreria i, de manera combinada amb la configuració, permet obtenir els resultats esperats.

5.7 Aplicació integrant la llibreria

Per a finalitzar el projecte i provar la seva utilitat, s'ha creat un petit joc que utilitza "C++ RPG Terrains". Això és útil també per a assegurar-nos que la llibreria té totes les eines necessàries per a la seva fàcil integració dins un projecte extern. Si bé al llarg del desenvolupament s'han anat comentant les funcionalitats de la llibreria amb altres programadors i s'han tingut en compte les seves opinions a l'hora de dissenyar la interacció amb la llibreria, no hi ha millor manera d'assegurar-nos que provar-ho i testejar.

Per exemple, si bé ja teníem una funció per a obtenir l'alçada i la normal del terreny en una certa posició, x i y, no teníem l'opció de fer "Raycast", per a detectar la intersecció entre el terreny i un raig infinit.

Aquest joc es tracta d'un petit joc d'avions, en què el jugador pot explorar i volar pels terrenys generats, buscant i donant caça a objectius generats de manera aleatòria al llarg de la superfície del terreny.



F 5.27 - Aplicació utilitzant C++ RPG Terrains

5.8 Tècniques utilitzades

A continuació, a les taules 5.2, 5.3, 5.4, 5.5, es pot veure un resum de les tècniques analitzades pel projecte: implementades, comentades com a treball futur o implementades i posteriorment substituïdes per un altre mètode més adequat.

Generació de soroll			
Técnica	*Implementada*	*Rebutjada*	*Treball futur*
Perlin noise bàsic		**X**	
Perlin noise fractal	**X**		
Ridged perlin noise	**X**		
Corba d'alçades	**X**		
Enviromental maps			**X**
Soroll infinit	**X**		
Generació al fil principal		**X**	
Generació en un fil paral·lel	**X**		
Generació a la GPU			**X**
Càlcul de normals a temps real		**X**	
Pre-calcular normals	**X**		

T 5.2 - Tècniques de generació de soroll analitzades

Generació de meshes			
Técnica	*Implementada*	*Rebutjada*	*Treball futur*
Mesh única per cada chunk		**X**	
Mesh única instanciada	**X**		
Alçades aplicades durant la generació		**X**	
Alçades aplicades durant el render	**X**		
Mesh d'alta resolució		**X**	
Subdivisió dinàmica	**X**		

T 5.3 - Tècniques de generació de meshes analitzades

Texturitzat i render			
Técnica	*Implementada*	*Rebutjada*	*Treball futur*
Textures condicionals	**X**		
Agrupar textures en una textura 3D			**X**
Texture displacement			**X**
Texture lineal blending		**X**	
Texture heightmap blending	**X**		
Mesh displacement through heightmap			**X**
Normal maps			**X**
Skyboxes			**X**
Boira	**X**		

T 5.4 - Tècniques de texturitzat i render analitzades

Nivells de detall (LODs)			
Técnica	*Implementada*	*Rebutjada*	*Treball futur*
Sets d'índexs		**X**	
Tessel·lació per distància		**X**	
Tessel·lació per mida en pantalla	**X**		
LOD stitching	**X**		

T 5.5 - Tècniques de LODs analitzades

6. Conclusions i treballs futurs

La primera fita assolida en aquest treball ha estat anàlisi en profunditat de les característiques dels terrenys en l'àmbit dels videojocs i allò que cal tenir present per a la seva generació a partir d'una llibreria desenvolupada amb aquest únic objectiu.

En videojocs, els terrenys no deixen de ser una malla poligonal 3D, o "mesh", com qualsevol altre element tridimensional. El que els diferencia de la resta d'objectes són les seves dimensions, sobretot en videojocs de mon obert. Poden arribar a tenir extensions quilomètriques, i ser visibles totes elles alhora, el que requereix moltes optimitzacions i desviacions del funcionament d'una malla normal per a ser sostenible. Algunes d'aquestes es discuteixen en aquest document, com la tessel·lació dinàmica o la instanciació d'un únic fragment de malla de manera repetida per a cada segment de terreny o "chunk".

A més a més, els terrenys tenen característiques comunes que permeten prescindir de certs elements o característiques que altres malles sí que poden tenir, com ara els volats o els forats, a canvi d'una millora significativa del rendiment, sumat a les optimitzacions mencionades anteriorment. Hi ha tècniques per a suplir aquestes limitacions, una de les més comunes per als terrenys basats en un heightmaps essent la col·locació de malles mesclades amb el terreny en sí, per a augmentar el detall, així com les diferents textures aplicables per a augmentar el detall i la complexitat de la superfície: displacement maps, normal maps, heightmaps, decals...

A més a més, en mesclar els terrenys amb la creació procedural apareixen nous reptes a sobrepassar: no sols la creació d'una eina que sigui capaç de generar el contingut procedural de manera estable, com el soroll perlin utilitzat en aquest projecte; sinó que generi contingut interessant per al jugador. Per aquest motiu és que s'han d'aplicar variacions i afegir complexitat al resultat, per a fer la impressió que aquella secció de contingut ha estat planejada i dissenyada amb cura, si no que tingui sentit dins les expectatives del jugador. En aquest cas, muntanyes en forma de muntanyes i amb relleu irregular, per les quals hem utilitzat afegits al perlin com el "ridged noise", els fractals i corbes d'alçada.

I, en aquest cas en particular, la generació del nou contingut ha de ser ràpida i dinàmica, doncs els terrenys s'han de generar en temps d'execució sense impactar el rendiment de l'aplicació on s'utilitzi la llibreria. És per això que certes tècniques, com la simulació d'erosió, han estat descartades, el temps que s'hi podria dedicar substituït per noves millores, com el multithreading de la llibreria.

A més a més, en tractar-se de la creació d'una llibreria, el codi exposat als usuaris ha de ser usable, clar, intuïtiu i funcional; així com la naturalesa procedural del projecte requereix que sigui personalitzable, dinàmic i eficient. Tot això són requeriments que han dut a la condensació de les funcionalitats en sols dos fitxers, per a la configuració del terreny i les accions que es poden dur a terme sobre el terreny, com es comenta a l'apartat 5.6, Interacció amb la llibreria.

La segona gran fita assolida ha estat el propi desenvolupament d'aquesta llibreria de generació de terrenys, a partir de les tècniques i solucions seleccionades a partir de l'estat de l'art.

Per a desenvolupar la llibreria és necessari un entorn de desenvolupament en què treballar, per a tota la funcionalitat externa a aquesta necessària per al funcionament de l'aplicació, el testeig i l'execució. Aquest codi extern s'encarrega de tot allò que no són els terrenys: la interacció amb l'usuari, la càrrega de textures, configurar l'estat dels terrenys.... En aquest cas, es tracta de "V_Engine" (31), una eina desenvolupada prèviament i que dóna el nom provisional a aquest projecte com a "V_Terrains", que posteriorment seria canviat a "C++ RPG Terrains".

Els terrenys usen com a base una textura 2D, un heightmap, generat a través de l'algoritme del soroll Perlin, que ens permet generar una textura pseudo-aleatòria: diferent per a cada llavor, però capaç de proporcionar sempre un mateix resultat per a una mateixa posició i inicialització amb la mateixa llavor. A més a més, se superposen variacions del mateix terreny sobre ell mateix de manera fractal per a afegir més variabilitat i interès al resultat, i apliquem la fórmula del "Ridged Perlin Noise", perquè aquest resultat sigui més natural, simulant serralades i valls.

A més a més, tota la generació del soroll es du a terme en un fil paral·lel al principal, de manera que els càlculs tenen un impacte menor en l'aplicació. És per això que en aquest fil es precalculen també les normals de tots els polígons per a poder ser aplicades directament a través del shader sense intermediaris.

Aquest heightmap és projectat sobre una malla, la generació de la qual ha passat per diferents etapes i mecanismes. Finalment, s'ha optat per generar una única malla de poca densitat poligonal i sense alçades, que és instanciada múltiples cops per a cada secció del terreny, o cada "chunk". Cada un d'aquests chunks té una textura de heightmap assignada, que es projectarà sobre la malla en el shader per a crear les alçades. A més a més, aquesta malla és subdividida a través d'un mètode de tessel·lació poligonal, segons la mida en pantalla de cada un dels triangles, per a poder afegir més detall al resultat final quan l'usuari s'hi acosti.

El shader, també s'encarrega de distribuir les textures adequadament. Aquestes tenen vinculades una sèrie de condicions que decideixen en quines parts del terreny poden i no poden aparèixer, i com s'han de mesclar entre elles a les fronteres.

Tots aquests "chunks", fragments de terreny, són gestionats per un "Chunk Manager", encarregat de sol·licitar la creació de nous chunks i d'esborrar els que estan massa distants del jugador per a ser visibles i han desaparegut ja dins la boira, calculada també en els shaders, per a ocultar els límits del terreny i simular la "infinitat" en extensió de la superfície.

Finalment, la tercera fita assolida ha estat la prova d'ús de la llibreria en un joc en que un avió sobrevola els terrenys. Aquest exercici ha demostrat l'eficàcia del desenvolupament i demostra la seva potencial aplicació real.

A continuació, a la taula 6.1, revisem els objectius mencionats a l'inici del document i el seu estat i la seva finalització, per a avaluar l'evolució del projecte.

O1. Dissenyar i desenvolupar una llibreria per a la generació dinàmica de terrenys Aquest objectiu ha estat assolit, ja que es tracta de l'objectiu més genèric d'aquest treball. La llibreria és funcional i és capaç de generar terrenys variats i relativament complexes a temps real i de manera dinàmica.
O2.Permetre a l'usuari incloure modificacions al funcionament de la llibreria Aquest objectiu ha variat lleugerament al llarg del desenvolupament. Si bé a l'inici s'enfocava com "la capacitat d'injectar codi dins la llibreria per a personalitzar-ne el funcionament". Aquesta aproximació, si bé s'ha dut a terme en punts específics com ara les corbes d'alçades, ha estat substituïda per a un mètode més clàssic, modificant el funcionament de la llibreria i la generació del terreny a través de variables i mètodes. L'objectiu segueix essent assolit, perquè la personalització i les eines segueixen a disposició de l'usuari, sobretot dins l'arxiu "config.h", mostrar a la figura 5.25.
O3. Aportar les eines necessàries per a incloure al terreny textures definides per l'usuari A través del sistema de textures condicionals, discutit a l'apartat 5.4.1, es permet que l'usuari utilitzi qualsevol textura i modifiqui la seva disposició sobre el terreny per a assolir resultats variats. Si bé el resultat obtingut, la qualitat de render no és tan alta com s'especifica a la taula 1.1, funcionalitats, aquest treball queda pel futur, doncs la llibreria te les bases necessàries per a poder obtenir aquests resultats. Modificar la malla segons el heightmap de la textura, PBR, skybox...
O4. Publicar la llibreria de manera oberta, sota la llicència GNU General Public License(29) La llibreria es troba de manera pública online, i sota la llicència GNU, com s'especifica a l'objectiu. Si bé encara no s'ha fet una divulgació intensiva ni s'ha publicat a més entorns que el seu repositori de github, és accessible per a tothom, i queda a l'espera de continuar millorant i desenvolupar les tasques marcades com a "treballs futurs" abans de divulgar més intensament la llibreria.
O5. Analitzar l'eficiència i rendiment dels mètodes emprats per a generar i renderitzar terrenys Molts dels mètodes rebutjats, especificats a la taula 5.2, tècniques utilitzades, han estat posats a banda a causa d'aquest objectiu. Molts d'ells eren lents, ineficients, o desenvolupats com a mètode temporal per a poder prosseguir amb la creació de la llibreria. I, gran part del desenvolupament de la llibreria s'ha centrat en aquest objectiu: des dels LODs i les seves diferents implementacions fins a l'aplicació de multithreading per a la generació de soroll.
O6. Crear una petita aplicació que utilitzi la llibreria per demostrar la seva utilitat L'aplicació ha estat creada i utilitza la llibreria de manera correcta.

T 6.1 - Avaluació dels objectius

Referent a la planificació temporal i les tasques i temps establerts a l'inici del desenvolupament, les desviacions ja s'han comentat a l'apartat 3.5. Les tasques de tessel·lació i texturitzat han durat més temps de l'esperat, ja que han aparegut moltes dificultats durant el seu desenvolupament. S'han provat moltes solucions diferents, amb diferents aproximacions, però totes elles tenien inconvenients diferents. No s'ha trobat una solució ideal, però sí la més adequada de totes. Tot i això, s'han acabat provant molts mètodes diferents, el que ha allargat la durada.

Durant el desenvolupament, moltes tasques han presentat una variabilitat més elevada de l'esperada en començar el projecte, pel que s'han avaluat i, en diverses ocasions, provar les diferents opcions per a obtenir un resultat concret, com és el cas dels nivells de detall (LOD), que han sofert moltes iteracions. Moltes de les iteracions i proves realitzades, si bé no apareixen en el resultat final, sí que es veuen reflectides en la qualitat obtinguda de la solució triada i la convicció de seguir el camí correcte, així com les noves tècniques apreses per als altres mètodes aplicades i/o mesclades amb la solució final.

Per altra banda, altres parts del treball, com ara la generació de soroll, si bé varen ser funcionals des de l'inici i la seva creació, però s'han anat modificant i millorant al llarg del desenvolupament, més enllà de la data marcada com a màxim per a treballar-hi: el "ridged perlin noise", l'optimització del thread de generació, l'emmagatzematge de la informació dins els chunks per a poder enviar aquesta informació a l'usuari quan es requereixi...

I d'altres, com el render, que si bé són funcionals, molts elements han quedat marcats com a "treballs futurs", doncs la qualitat sempre és millorable, i el rendiment sempre pot ser optimitzat.

En general, el món dels terrenys és complex i profund i amb la durada d'aquest treball, de 300 hores (teòriques), s'ha aconseguit desenvolupar un prototip més que un resultat comercial i/o final. Amb temps, es podria obtenir un producte acabat, més adequat a les necessitats comercials d'avui en dia. Tot i això, com a resultat de recerca i de provar noves aproximacions a problemes coneguts, aquest treball ha estat un entorn ideal.

El resultat d'aquest projecte, C++ RPG Terrains, és públic sota llicència GNU (General Public License)(29) al repositori de Github "V_Terrains"(32), el nom previ que va rebre el projecte.

7. Bibliografia

1 Rezvan Mahdavi-Hezave and Raman Ramsin, *FDMD: Feature-Driven Methodology Development*, 2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), Barcelona, Spain, 2015

2 Julian Togelius, Emil Kastbjerg, David Schedl and Georgios N. Yannakakis, *What is Procedural Content Generation? Mario on the borderline*, Proceedings of the 2nd International Workshop on Procedural Content Generation in Games Article No. 3 Bordeaux, France. 2011 DOI 10.1145/2000919.2000922

3 Jack Copeland, Jonathan Bowen, Mark Sprevak, Robin Wilson and others, *The Turing Guide*, 2017

4 Ruben M. Smelik, Klaas Jan de Kraker and Saskia A. Groenewegen, *A Survey of Procedural Methods for Terrain Modelling*, Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS), 2009

5 Jacob Olsen, *Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games*, 2004

6 Houssam Hnaidi, Eric Guérin, Samir Akkouche, Adrien Peytavie and Eric Galin, *Feature based terrain generation using diffusion equation*, doi.org/10.1111/j.1467-8659.2010.01806.x, 2010

7 Michael Hesse and Marina L. Gavrilova, *An Efficient Algorithm for Real-Time 3D Terrain Walkthrough*, Conference: Computational Science and Its Applications - ICCSA 2003, DOI: 10.1007/3-540-44842-X_76, 2003

8 Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich and Mark B. Mineev-Weinstein, *ROAMing Terrain: Real-time Optimally Adapting Meshes*, Proceeding: VIS '97 Proceedings of the 8th conference on Visualization '97 Pages 81-88, 2002

9 Matthew White, *Real-Time Optimally Adapting Meshes: Terrain Visualization in Games*, dx.doi.org/10.1155/2008/753584, 2007

10 Guojun Chen and Jing Zhang, *Dynamic Terrain LOD with Region Preservation in 3D Game Engine*, Lecture Notes in Computer Science book series (LNCS, volume 3942), 2006

11 Filip Strugar, *Continuous Distance-Dependent Level of Detail for Rendering Heightmaps*, DOI:10.1080/2151237X.2009.10129287, 2011

12 Daniel Sánchez-Crespo Dalmau, *Core Techniques and Algorithms in Game Programming*, ISBN-13: 978-0131020092, 2004

13 Tuomo Hyttinen, *Terrain syhnthesis using noise*,Proceeding AcademicMindtrek '17 Proceedings of the 21st International Academic Mindtrek Conference, 2017

8. Webgrafia

14 Agile methodology: Feature Driven Development
- agilemodeling.com/essays/fdd.htm

15 Rikarts
- citmproject3.github.io/Project3

16 HackNPlan
- hacknplan.com

17 Types of terrains
- gamedev.stackexchange.com/questions/15573/heightmap-voxel-polygon-geometry-terrains

18 Procedural Content Generation (PCG)
- davideaversa.it/wp-content/uploads/2015/06/Procedural-Contents-Generation.pdf

19 VTerrain.org
- vterrain.org

20 Understanding perlin noise - Adrian's soapbox
- flafla2.github.io/2014/08/09/perlinnoise.html

21 Noise types & Diamond-square algorythm
- jayelinda.com/modelling-by-numbers-supplementary-terrain

22 Procedural Landmass Generation Tutorial in Unity3D
- youtu.be/wbpMiKiSKm8

23 Unity
- unity3d.com

24 Unreal
- www.unrealengine.com

25 World Creator
- world-creator.com

26 World Machine
- world-machine.com

27 PicoGen
-picogen.org

28 Terrain Party
- terrain.party

29 GNU General Public License
- www.gnu.org/licenses/gpl-3.0.en.html

30 Speed Tree
- store.speedtree.com

31 V_Engine
- github.com/Vulpem/V_Engine

32 V_Terrains / C++ RPG Terrains Github repository
- github.com/Vulpem/V_Terrains